Global Modeling Initiative: Tutorial and User's Guide II

Jules Kouatchou, Megan Damon and Gary Wojcik
Software Integration and Visualization Office
NASA Goddard Space Flight Center
Greenbelt, MD 20771

March 13, 2008

Abstract

In this report, we provide a description of the GMI code. It is intended to help users in the GMI community to obtain, install, compile, run, and modify the code. We present the code organization and data structures, procedures on how to run the code under various configurations, to manipulate the code, and the code parallel performance. Examples introduced here were carried out on Intel clusters.

This version of the User's Guide differs from the previous one (released in 2004) in many areas due to the fact that the GMI code went through a lot of changes. For instance:

- A completely new code directory structure
- Componentization of the code
- New namelist settings
- New diagnostics capabilities.
- Removal of the LLNL ESM package
- Introduction of ESMF.

As a result of these changes, we have new installation and compilation procedures, an object-oriented approach to manipulate components of the code, a more flexible way to produce netCDF output file, a more readable code, the ability to automatically generate code documentation, etc.

Contents

1 Introduction **1**

2 Structure of the Code **3**
2.1 Components of GMI 3
2.2 Directory Structure of The Code 4
2.3 Coding Principles 8
2.4 General Flowchart of the Code 8

3 Installation and Testing **11**
3.1 Getting the Code 12
3.2 Model Files and Directory Structure 12
3.3 Setting Environment Variables 12
3.4 Code Compilation and Basic Test Run 14
3.4.1 Compiling the model 14
3.4.2 Testing the Executable 14
3.5 Summary of the Necessary Steps 16

4 GMI Files **17**
4.1 Input Namelist Files 17
4.2 Input Datasets 18
4.3 Output Files 21
4.3.1 ASCII Diagnostic Output File 21
4.3.2 netCDF Output Files 22
4.4 Location of Input, Output, and Other GMI Files 23
4.4.1 Directories under input/ 23
4.4.2 Directories under output/ 24

5 Performing Specific Runs **25**
5.1 The GMI Self Contained Test 25
5.2 Standard Runs 25
5.3 Other Namelist Variables 26
5.4 Restart Capabilities 26

6 Making Changes **29**
6.1 Coding Consideration 29
6.2 Making Changes in the Code 29
6.2.1 Adding a Variable to a Derived Type 31
6.2.2 Adding Chemical Mechanisms 32
6.2.3 netCDF Output Files 33

7 Script Tools **34**
7.1 Internal Scripts 34
7.2 Production Tools 36

8 How to Use CVS **39**
8.1 What is CVS? 39
8.2 How to Use CVS 40
8.3 Use CVS to Keep Up-to-Date with GMI Source Code Changes 40
8.4 Use CVS to Track Both New Releases and Your Changes 43
8.5 Where To Obtain CVS 44

9 Parallel Performance **45**

A Include Files **49**

B Single/Multiple Processor Runs **51**

C NetCDF Files **53**

D Important Features **58**
D.1 From Species Indices to Species Names 58
D.2 Station Diagnostics 61
D.3 Frequency Diagnostics 63
D.4 Overpass Diagnostics 64
D.5 Choice of Vertical Levels 65

E List of Species **66**

F Input Namelist Variables **70**

List of Tables

2.1 Basic information on the chemical mechanisms. 4
2.2 GMI code directories 7

4.1 List of namelist variables refering to input file names. The chemical mechanisms are labeled as follow: aerosol (A), gocart_aerosol (GA), micro_aerosol (MA), troposphere (T), and strat_trop (C). 20
4.2 Available metFields files. 21

C.1 GMI netCDF files. Here # is 1, 2, 3, or 4. 53
C.2 Content of each GMI netCDF output file. 56
C.3 Frequency of GMI netCDF output files. 56

D.1 New namelist variables (and corresponding old ones) used to set species names instead of species indices in the namelist file. 59
D.2 New namelist variables for station diagnostics. 61

F.1 Namelist variables 88

List of Figures

2.1 Flowchart of the main program 9
2.2 Flowchart of the time stepping routine. 10

Chapter 1

Introduction

The Global Modeling Initiative (GMI) [1] was initiated under the auspices of the Atmospheric Effects of Aircraft Program (AEAP) in 1995. The goal of GMI is to develop and maintain a state-of-the-art modular 3-D chemistry transport model (CTM) that can be used to assess the impact of various natural and anthropogenic perturbations on atmospheric composition and chemistry, including, but not exclusively, the effect of aircraft.

The Atmospheric Chemistry Modeling and Analysis Program (ACMAP) has selected the approach of GMI to serve both as an assessment facility and a testbed for model improvements for future assessment in all areas of atmospheric chemistry. The goals in the design of GMI as an assessment tool are [4]

1. The model should be well-characterized and thoroughly tested against observations.

2. The model should be able to test and compare a diversity of approaches to specific processes by being able to easily swap modules containing different formulations of chemical processes, within a common framework.

3. The model should be optimized for computational efficiency and be able to run on different platforms.

4. Model results should be examined by a large representation of the scientific community, thus faciliting consensus on the significance of assessment results.

5. Ultimately, the model integration could provide a unique assessment capability for other anthropogenic impacts of concern by providing a testbed for other algorithms and intercomparisons used in assessment of those issues.

Many elements of the GMI model address these goals. The GMI model is a modular chemistry-transport model (CTM) with the ability to carry out multi-year assessment simulations as well as incorporate different modules, such as meteorological fields, chemical mechanisms, numerical methods, and other modules representing the different approaches of current models. This capability facilitates the understanding of the differences and uncertainties of model results.

[1] http://gmi.gsfc.nasa.gov/gmi.html

The testing of GMI results against observations is a high priority of GMI activities. Science Team members contribute by either supplying a particular module and/or contributing to the analysis of the results and comparison with atmospheric observations [3, 6, 7, 5]. Application of the model to the potential impacts of stratospheric aircraft emissions is presented in [4]. The model has been employed to investigate the effects of stratospheric aircraft emissions on polar stratospheric clouds [2] and simulate ozone recovery over a 36-year time period [1].

Besides acting as a testbed for different modules, GMI will also act as a 3-D assessment facility. The GMI modular code is currently implemented at NASA/Goddard Space Flight Center (the core institution). The core institution is responsible for:

- Integrating and testing components of the GMI model,
- Making the code readable, flexible and easy to maintain
- Maintaining coding standards which will make the model portable to different platforms
- Carrying out assessment calculations, and
- Providing first-order results and diagnostics for analysis by team members.

The current version of the code has been developed to run on a variety of computing platforms, both with single and multiple processors (Linux clusters, SGI Origin series, HP Compaq SC45, single processor workstations, etc.).

This report is intended to familiarize users with the GMI code. Users will be able to

- Have information on the code structure (Chapter 2).
- Obtain instructions on how to obtain the code, install it, compile it, run it on any platform (Chapter 3).
- Have knowledge of all the input and output files involved in the code (Chapter 4, Appendix C and Appendix F).
- Carry out specific and restart runs (Chapter 5).
- Learn how to make changes in the code (Chapter 6).
- Execute useful script tools needed, for instance, to search for words, to produce restart input namelist files, etc. (Chapter 7).
- Learn basic CVS commands (Chapter 8).
- Analyze the parallel performance of the code (Chapter 9).
- Be familiar with include files used to select the desired architecture, to set up compilation options, etc. (Appendix A).
- Know how to carry out a single or multiple processor run (Appendix B).
- Use namelist features (Appendix D).
- Know the species used in the code (Appendix E).

Chapter 2

Structure of the Code

2.1 Components of GMI

The modules that make up the GMI assessment model are [6]:

1. Input meteorological data coming from four major Global Circulation Models (from NCAR, GISS, DAO, and GMAO). Data from all these input sets include horizontal U and V winds, temperature, and surface pressure.
2. Advection algorithm to transport trace species
3. Mass tendencies
4. Numerical schemes for chemistry solutions
5. Chemistry mechanism
6. Heterogeneous processes
7. Photolysis
8. Diagnostics
9. Tropospheric treatment
10. Initial conditions
11. Boundary conditions

All the above modules have multiple packages/versions that can be selected through proper namelist settings. The GMI model incorporates six chemical mechanisms:

- aerosol (University of Michigan formulation)
- micro_aerosol (University of Michigan formulation)
- gocart_aerosol (GOCART formulation)
- stratosphere

- troposphere
- combined stratostophere/troposphere (strat_trop or combo)

A summary of the mechanisms appears in Table 2.1.

Mechanism name	# species	# thermal reactions	# photolytic reactions
aerosol	30	8	1
micro_aerosol	40	8	1
gocart_aerosol	31	8	1
stratosphere	57	122	44
troposphere	85	222	49
strat_trop	124	320	81

Table 2.1: Basic information on the chemical mechanisms.

2.2 Directory Structure of The Code

The top directory of the GMI code is *gmi_gsfc/* which contains the sub-directories

- *Components/*: Routines for each component
- *Shared/*: Include files and routines shared by all the components
- *Documents/*: General information about the code and how it is to be used
- *Applications/*: Routines for driving the code
- *Config/*: Main makefile files

In Table 2.2, we give more details on the structure of each of the above directories.

Directory Name	Synopsis
gmi_gsfc	GMI reference directory
gmi_gsfc/Documents	Directory for documents
gmi_gsfc/Documents/Papers	
gmi_gsfc/Documents/Tutorials	
gmi_gsfc/Documents/Tutorials/UserGuide	This document
gmi_gsfc/Documents/Tutorials/removingESM	
gmi_gsfc/Documents/Tutorials/MEGANemissions	
gmi_gsfc/Documents/ReadmeFiles	
gmi_gsfc/Config	Main makefile files
gmi_gsfc/Shared	Files and modules shared by all the components
gmi_gsfc/Shared/GmiCommunications	Communication routines
gmi_gsfc/Shared/GmiIOutilities	Supporting routines for I/O
gmi_gsfc/Shared/GmiESMF	Interfaces to ESMF
gmi_gsfc/Shared/GmiInclude	Include files
gmi_gsfc/Shared/GmiMetFields	Routines for deriving MetFields variables
gmi_gsfc/Shared/GmiScripts	Script tools
gmi_gsfc/Shared/GmiSupportingModules	Supporting routines for various calculations.
gmi_gsfc/Shared/NcUtils_Double	netCDF utility routines for double precision
gmi_gsfc/Shared/NcUtils_Single	netCDF utility routines for single precision
gmi_gsfc/Components/GmiAdvection	Advection component
gmi_gsfc/Components/GmiAdvection/advectionMethod	Routines driving Advection
gmi_gsfc/Components/GmiAdvection/dao2advec	DAO advection routines
gmi_gsfc/Components/GmiAdvection/dao2utils	Advection utility routine computing, courant numbers, Divergence, etc.
gmi_gsfc/Components/GmiAdvection/include	Advection include file
gmi_gsfc/Components/GmiChemistry	Chemistry component
gmi_gsfc/Components/GmiChemistry/chemistryMethod	Routines driving Chemistry
gmi_gsfc/Components/GmiChemistry/AerosolDust	Module for Aerosol/Dust calculations
gmi_gsfc/Components/GmiChemistry/ioChemistry	I/O routines for Chemistry
gmi_gsfc/Components/GmiChemistry/include	Chemistry include file
gmi_gsfc/Components/GmiChemistry/sad	Aerosol surface area density and condensed phase mixing ratio modules

gmi_gsfc/Components/GmiChemistry/solvers	Chemistry solvers
gmi_gsfc/Components/GmiChemistry/solvers/micro_sulfur	
gmi_gsfc/Components/GmiChemistry/solvers/smv2chem	
gmi_gsfc/Components/GmiChemistry/solvers/sulfur	
gmi_gsfc/Components/GmiChemistry/mechanisms	Chemical mechanisms
gmi_gsfc/Components/GmiChemistry/mechanisms/aerosol	Aerosol chemistry
gmi_gsfc/Components/GmiChemistry/mechanisms/aerosol/include_setkin	Include files for aerosol
gmi_gsfc/Components/GmiChemistry/mechanisms/aerosol/setkin	Routines for rate constants and kinetic rates
gmi_gsfc/Components/GmiChemistry/mechanisms/micro_aerosol	micro_aerosol chemistry
gmi_gsfc/Components/GmiChemistry/mechanisms/micro_aerosol/include_setkin	Include files for micro_aerosol
gmi_gsfc/Components/GmiChemistry/mechanisms/micro_aerosol/setkin	Routines for rate constants and kinetic rates
gmi_gsfc/Components/GmiChemistry/mechanisms/gocart_aerosol	gocart_aerosol chemistry
gmi_gsfc/Components/GmiChemistry/mechanisms/gocart_aerosol/include_setkin	Include files for gocart_aerosol
gmi_gsfc/Components/GmiChemistry/mechanisms/gocart_aerosol/setkin	Routines for rate constants and kinetic rates
gmi_gsfc/Components/GmiChemistry/mechanisms/strat_trop	Strat/Trop chemistry
gmi_gsfc/Components/GmiChemistry/mechanisms/strat_trop/include_setkin	Include files for the combined strat/trop
gmi_gsfc/Components/GmiChemistry/mechanisms/strat_trop/setkin	Routines for rate constants and kinetic rates
gmi_gsfc/Components/GmiChemistry/mechanisms/stratosphere	Stratospheric chemistry
gmi_gsfc/Components/GmiChemistry/mechanisms/stratosphere/include_setkin	Include files for stratosphere
gmi_gsfc/Components/GmiChemistry/mechanisms/stratosphere/setkin	Routines for rate constants and kinetic rates
gmi_gsfc/Components/GmiChemistry/sulfur	Routines for sulfur chemistry
gmi_gsfc/Components/GmiChemistry/mechanisms/troposphere	Tropospheric chemistry
gmi_gsfc/Components/GmiChemistry/mechanisms/troposphere/include_setkin	Include files for troposphere
gmi_gsfc/Components/GmiChemistry/mechanisms/troposphere/setkin	Routines for rate constants and kinetic rates
gmi_gsfc/Components/GmiChemistry/photolysis	Photolysis component
gmi_gsfc/Components/GmiChemistry/photolysis/include	
gmi_gsfc/Components/GmiChemistry/photolysis/fastj	
gmi_gsfc/Components/GmiChemistry/photolysis/fast_JX	
gmi_gsfc/Components/GmiChemistry/photolysis/fast_JX53b	
gmi_gsfc/Components/GmiChemistry/photolysis/fast_JX53c_ref	
gmi_gsfc/Components/GmiChemistry/photolysis/lookup	Routines for lookup table
gmi_gsfc/Components/GmiChemistry/photolysis/utils	
gmi_gsfc/Components/GmiConvection	Convection component
gmi_gsfc/Components/GmiConvection/convectionMethod	Routines driving Convection

gmi_gsfc/Components/GmiDeposition	Deposition component
gmi_gsfc/Components/GmiDeposition/depositionMethod	Routines driving Deposition
gmi_gsfc/Components/GmiDeposition/include	
gmi_gsfc/Components/GmiDiffusion	Diffusion component
gmi_gsfc/Components/GmiDiffusion/diffusionMethod	Routines driving Diffusion
gmi_gsfc/Components/GmiEmission	Emission component
gmi_gsfc/Components/GmiEmission/emissionMethod	Routines driving Emission
gmi_gsfc/Components/GmiEmission/Harvard	Harvard emission routines
gmi_gsfc/Components/GmiEmission/MEGAN	MEGAN emission routines
gmi_gsfc/Components/GmiEmission/ioEmission	I/O routines for Emission
gmi_gsfc/Components/GmiEmission/include	
gmi_gsfc/Components/GmiEmission/lightning	Routines for lightning parameterization
gmi_gsfc/Components/GmiEmission/llnl	LLNL emission routines
gmi_gsfc/Components/GmiSpeciesConcentration	Species Concentration component
gmi_gsfc/Components/GmiSpeciesConcentration/spcConcentrationMethod	Routines driving Species Concentration
gmi_gsfc/Components/GmiSpeciesConcentration/ioSpcConcentration	I/O routines for Species Concentration
gmi_gsfc/Applications	Control and time stepping routines
gmi_gsfc/Applications/GmiApp	
gmi_gsfc/Applications/GmiBin	

Table 2.2: GMI code directories

2.3 Coding Principles

A ".F90" (Fortran code) suffix denotes source code files. The "F90" suffix indicates that the Fortran source contains preprocessing directives. Files named with a ".h" suffix are header files that contain preprocessing directives, variable declarations, parameter definitions, and common block definitions. Contents of selected header files are included via *#include* statements at the beginning portion of each of the ".F90" and ".c" files.

To enable multiyear chemistry simulations, the GMI core model was parallelized to make use of the most powerful computational platforms available. The parallel strategy uses a two-dimensional longitude/latitude domain decomposition whereby each subdomain consists of a number of contiguous columns having a full vertical extent. Processors are assigned to subdomains, and variables local to a given subdomain are stored on the memory of the assigned processor. Data are transmitted between computational processes, when needed, in the form of messages. The number of meshpoints per subdomain may not be uniform, under the constraint that the decomposition be logically rectangular. The choice to decompose in only two dimensions is based on the fact that chemistry, photolysis, and cold sulfate algorithms make up the vast majority of the computational requirements and are all either local or column calculations. These computations require no communication with neighboring grid zones and hence maximize the parallel efficiency [6].

2.4 General Flowchart of the Code

In this section, we provide flowcharts of the main program and the time stepping routine. The main program is *gmiMain* (filename gmiMain.F90). This calls routines to initialize the ESMF programming environment (that launches the Message Passing Interface MPI), perform domain decomposition, initialize the GMI components (Chemistry, Emission, Deposition, Advection, Convection, and Diffusion), perfom model integration (run the components), and finalize the components and ESMF (see Figure 2.1). The most important calculations done in the time stepping routine appear in Figure 2.2. The figure only shows the main modules of the code. We observe that the major components are executed here.

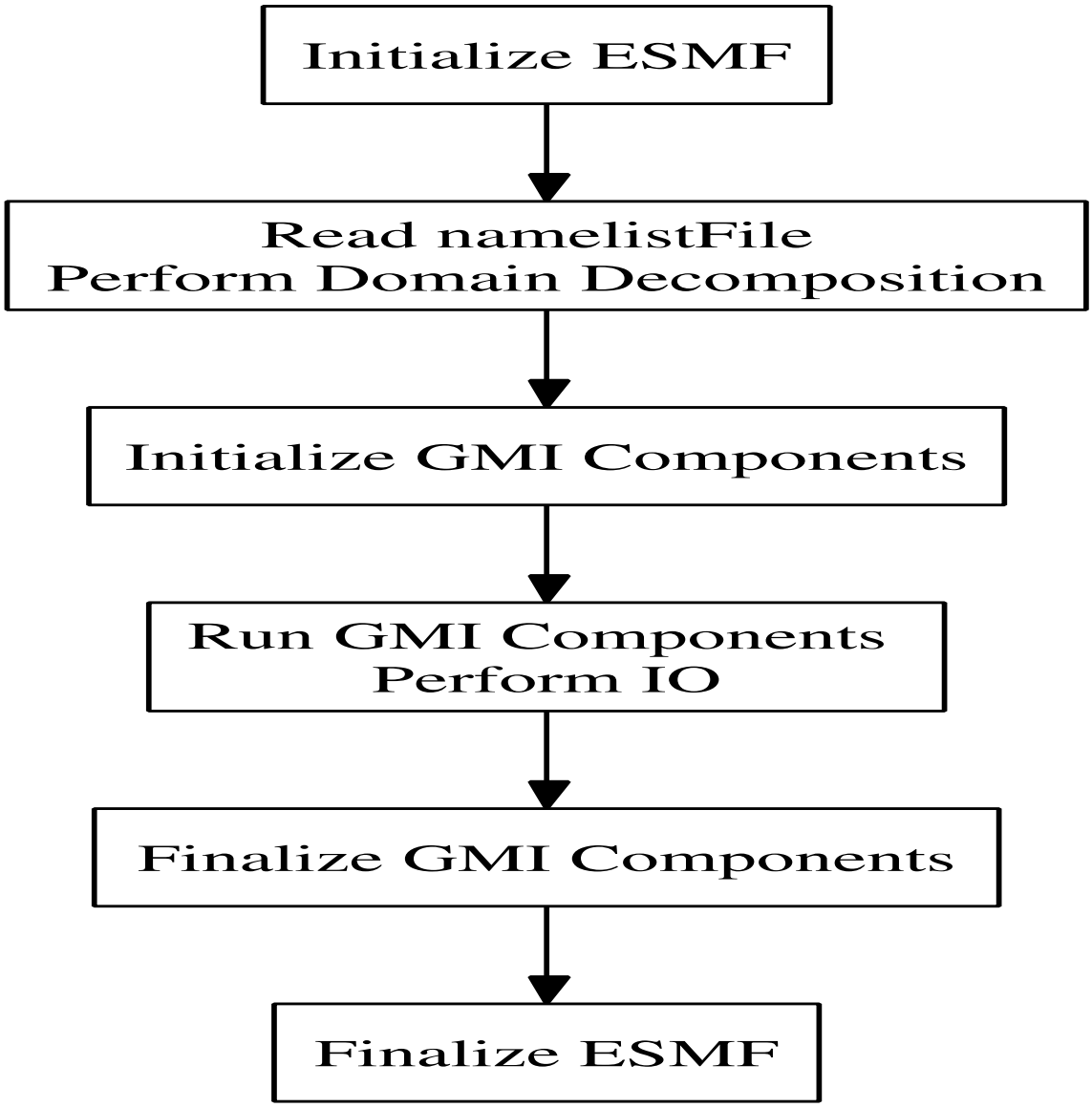

Figure 2.1: Flowchart of the main program

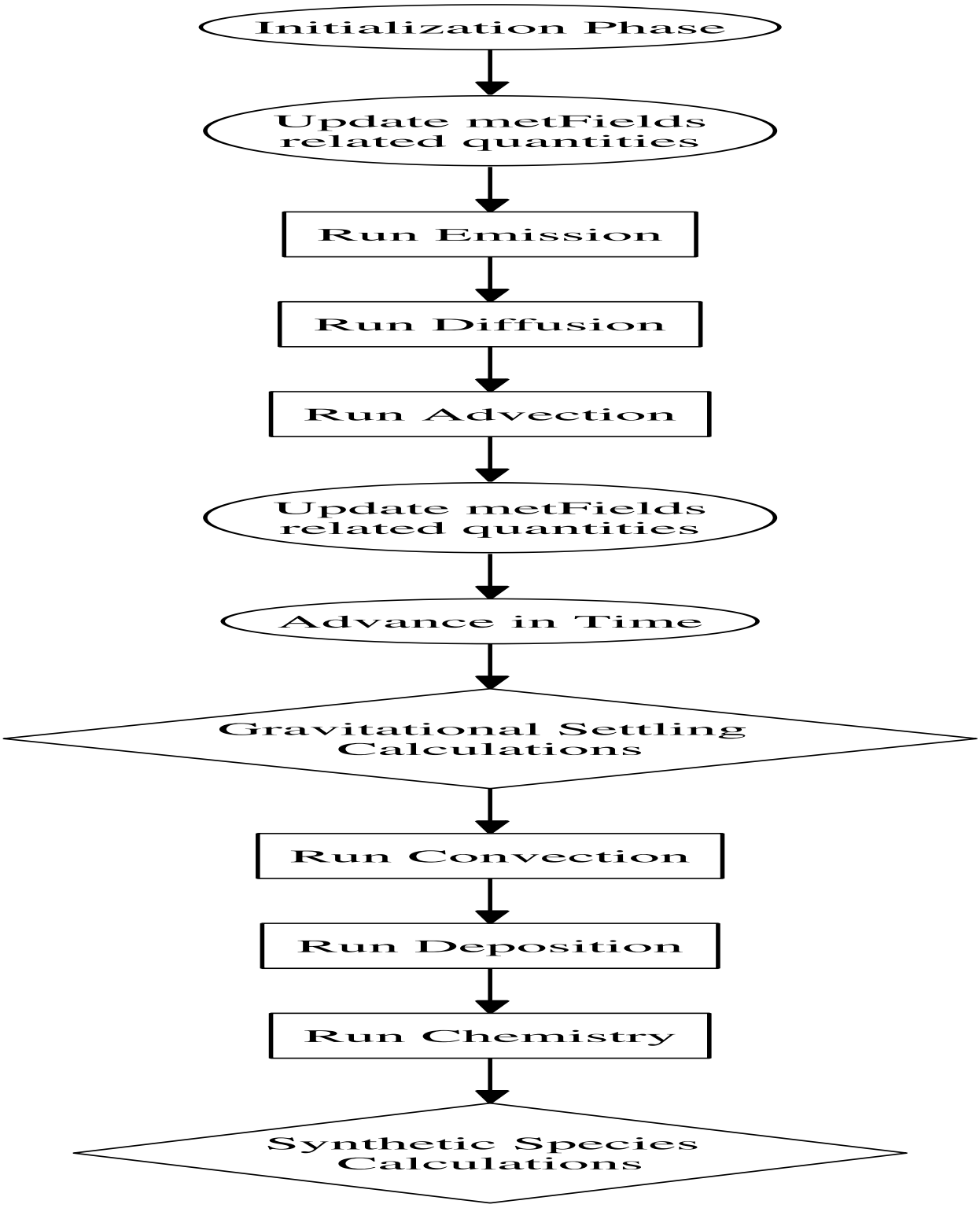

Figure 2.2: Flowchart of the time stepping routine.

Chapter 3

Installation and Testing

This chapter is written to help new users install and test the GMI code. We provide specific instructions on how to obtain the code, to properly set environment variables, to select the model configuration, to choose a particular platform, to compile the code and to perform basic test runs. The focus of these instruction is on the installation and execution of the GMI code on `discover` and `explore`. The same procedures can easily be applied to any platform.

To get and install the GMI code, the following system software is needed:

- CVS (see Chapter 8 for instruction)
- F90/95 (ideally *ifort* for intel)
- C (ideally *icc* for intel). Down the road, it will not be required.
- MPI
- netCDF (version 3.4 or higher). The location of netCDF should be provided in the files *Config/gem_config.h* and *Config/compiler.mk* before compiling the code (see Section 3.3 for details).
- ESMF (see `http://www.esmf.ucar.edu`)
- make and gmake
- makedepend (generally in /usr/bin/X11)
- perl
- a debugger (if possible)

During this process of installing and testing the code, it is assumed that *Cshell* is the default shell employed by the user. In fact, the GMI environment variables required for these procedures are set up using *Cshell*.

3.1 Getting the Code

To obtain the GMI code,

- Select the directory where you want to install the GMI model, say *MYGMI/*
- Get the latest version of the model from the cvs repository at *sourcemotel* by typing the command lines:

```
%setenv CVS_RSH ssh
%cvs -d usrid@sourcemotel.gsfc.nasa.gov:/cvsroot/gmi co -P gmi_gsfc
```

Here *usrid* is your login name on `sourcemotel`. You will be asked to provide your password on `sourcemotel`. The directory *gmi_gsfc/*, which is the main GMI directory, is then created.

3.2 Model Files and Directory Structure

Move into *gmi_gsfc/*, the top directory of the GMI code:

```
%cd gmi_gsfc
%ls
```

You will find (in *gmi_gsfc/*) the files and directories:

```
Applications/     Config/        README.first      Shared/       login.gmi
CVS/              Documents/     README.install    cshrc.gmi
Components/       Makefile       README.notice     gem/
```

3.3 Setting Environment Variables

In the directory *gmi_gsfc/*, read all the README files by starting with *README.first* file that guides a new user to take the required steps for installing and running the GMI code. The top portions of the files *cshrc.gmi* and *login.gmi* (also located in *Shared/GmiScripts/*) include instructions for setting up the environment variables which are discussed in this section.

Edit the file *cshrc.gmi*

- Select the chemistry mechanism you want to consider by setting the variable *CHEMCASE*. Currently six mechanisms are available: *troposphere*, *aerosol*, *micro_aerosol*, *gocart_aerosol*, *stratosphere*, and *strat_trop* (for the combined stratosphere/troposphere or combo mechanism). If you want to use the *troposphere* mechanism for instance, uncomment the corresponding line to have

```
setenv CHEMCASE trop
```

- For the platform you want the GMI model to run on, update the variables *GMIHOME* (location of the main model directory) and *GMI_DATA* (directory where the input data to test the installation are located - not really necessary):

```
setenv GEMHOME   ~/MYGMI/gmi_gsfc
setenv GMI_DATA  ~/MYGMI/gmi_gsfc
```

Copy the files *cshrc.gmi* and *login.gmi* from this directory to your home directory

```
%cp cshrc.gmi ~/.cshrc.gmi
%cp login.gmi ~/.login.gmi
```

Go to the directory *Config/* and edit the file *gem_sys_options.h*. Modify the line

```
#define ARCH_OPTION  ARCH_XXXX
```

to select the architecture on which you want to run the code. For instance, *XXXX* is *INTEL* for `discover` or `explore`. In this case, you also need to set

```
#define HOST_MACH  YYYYY
```

where *YYYYY* is either *DISCOVER* or *PALM* (for *explore*).
In addition, set the variable *MSG_OPTION* to determine if you want a single processor version of the code

```
#define MSG_OPTION    MSG_NONE
```

or a multiple processor version (using MPI) of the code

```
#define MSG_OPTION    MSG_MPI
```

You may also choose to edit the file *gem_options.h* to select debugging, optimization, or profiling options. Provide the paths to MPI and netCDF include files and libraries in the files *gem_config.h* and *libraries.mk*. Some compilation options may have to be changed in the files *gem_config.h* and *compiler.mk*. Go to your home directory and edit the file *.cshrc*

```
%cd ~/
%vi .cshrc
```

Include the lines

```
setenv CVS_RSH ssh
setenv ARCHITECTURE  ARCH_XXXX
if (-e ~/.cshrc.gmi) then
   source ~/.cshrc.gmi
endif
```

You must also edit the *.login* file and add the lines

```
if (-e ~/.login.gmi) then
    source ~/.login.gmi
endif
```

Update the changes made in the files *.cshrc* and *.login* by typing:

```
%source .cshrc
%source .login
```

The setting of the environment variables ended with the previous two commands. The setting automatically creates aliases that allow the user to easily access the code directories and to execute scripts (see Chapter 7). For instance, by typing:

- `cd gmi` or `cd $GMIHOME`, you will get to the code main directory.
- `cd phot`, you will move to the directory containing the Photolysis component package (*gmi_gsfc/Components/GmiChemistry/photolysis/*).
- `cd emiss`, you will move to the directory containing the Emission component package (*gmi_gsfc/Components/GmiEmission/*).
- `seabf my_words`, you will search through all the GMI ".F90" files for the string *my_words*.

3.4 Code Compilation and Basic Test Run

Go back to the working directory *MYGMI/gmi_gsfc/*:

```
%cd gmi
```

3.4.1 Compiling the model

To compile the code, use the commands:

```
%gmake all
```

The *gmake* command compiles and links the code. ".f90", ".o", ".mod" and ".a" files are created and the executable named *gmi.x* is placed in the directory *Applications/GmiBin/*.

3.4.2 Testing the Executable

To test the executable, we will use a sample namelist file that comes with the code. For each platform, we show examples of job script files (named *gmitest.job*) to test the executable. On `explore` and `discover` you need to have your sponsor code account (type the command `getsponsor` to obtain it).

It is assumed that the user wants to test the model from the directory */explore/nobackup/usrid* on `explore` and */discover/nobackup/usrid* on `discover`.

```
runGmi_ExploreTest
      #PBS -S /bin/csh
      #PBS -N gmiCombo
      ##     -N sets job's name
      #PBS -l ncpus=64
```

```
      #PBS -l walltime=00:35:00
      #PBS -A a930b
      ##      -A sets the sponsor code account
      #PBS -V
      #PBS -e gmiCombo.err
      #PBS -o gmiCombo.out
      #
      setenv workDir /explore/nobackup/usrid
      setenv CHEMCASE strat_trop
      setenv GmiBinDir ~/MYGMI/gmi_gsfc/Applications/GmiBin

      cd $workDir
      #
      mpirun -np 64 $GmiBinDir/gmi.x -d comboNamelistFile.in

runGmi_DiscoverTest
      #PBS -S /bin/csh
      #PBS -N gmiCombo
      ###    -N sets job's name
      #PBS -l select=16:ncpus=4
      ###     choose 16 nodes and 4 processors per node
      #PBS -l walltime=00:35:00
      #PBS -W a930b
      ###     -W sets the sponsor code account
      #PBS -V
      #PBS -e gmiCombo.err
      #PBS -o gmiCombo.out
      #
      setenv workDir /discover/nobackup/usrid
      setenv CHEMCASE strat_trop
      setenv GmiBinDir ~/MYGMI/gmi_gsfc/Applications/GmiBin

      cd $workDir
      #
      limit stacksize unlimited
      mpirun -np 64 $GmiBinDir/gmi.x -d comboNamelistFile.in
```

Remark 1 *Replace a930b in the above script files with your sponsor code account. Here* usrid *is the user's login name.*

To submit the job script, do the following

```
On explore and discover
    %qsub runGmi_ExploreTest
    %qsub runGmi_DiscoverTest
```

3.5 Summary of the Necessary Steps

In this section, we summarize the steps needed to obtain, install, and run the GMI code on any platform.

1. Obtain the code (*gmi_gsfc* release) from the cvs repository.
2. Move to the GMI working directory (*gmi_gsfc/*).
3. Edit the file *cshrc.gmi* to update the variables `GMIHOME`, `GMI_DATA` (not necessary) and `CHEMCASE`.
4. Copy the files *cshrc.gmi* and *login.gmi* to *.cshrc.gmi* and *.login.gmi* in your home directory.
5. Go to the directory *Config/* to edit the files *gem_sys_options.h*, *gem_config.h*, *compiler.mk*, and *libraries.mk* to select the architecture and to update the compilation options and paths.
6. Go to your home directory to edit and source the files *.cshrc* and *.login*.
7. Type `cd gmi` and compile the code by typing `gmake all`.
8. Write a job script file and submit the job to run the executable.

Chapter 4

GMI Input/Output Files

4.1 Input Namelist Files

Input namelist files are generally named <problem_name>.in. They allow many variables to be changed without having to recompile or relink the code. Each namelist file is broken into these sections:

1. nlGmiControl
2. nlGmiMetFields
3. nlGmiSpeciesConcentration
4. nlGmiTracer
5. nlGmiDiagnostics
6. nlGmiRestart
7. nlGmiAdvection
8. nlGmiConvection
9. nlGmiDiffusion
10. nlGmiDeposition
11. nlGmiEmission
12. nlGmiLightning
13. nlGmiChemistry
14. nlGmiPhotolysis

Here are some basic requirements for editing namelist files:

- All namelist sections must be present and in proper order, even if no variable are listed in them.

- Variable names must be exact and placed in the proper section.
- Real numbers need a "d" exponent, even if it is "d0".
- Put a comma after each variable entry except the last one in each section.
- End each section with a "/".
- File names must be enclosed in quotes.
- Some variable settings are incompatible with each other. The code does some checking operations to catch these at run time.

Remark 2 *All REAL*8 namelist variables need to be input with a "D" exponent (even if it is D0).*

Appendix F gives a list of all the namelist variables, their types, and their description.

4.2 Input Datasets

In Table 4.1, we list all the namelist variables for necessary input data and the file in which they are contained. We also briefly describe each file, state whether it is grid dependent, and indicate which chemical mechanism requires it, if any.

Variable Name	Type	Description	Grid Dep.	Mechanism
nlGmiSpeciesConcentration:				
const_infile_name	nc	Initial species concentration	yes	all
fixed_const_infile_name	nc	fixed species concentration	yes	all
nlGmiDiagnostics:				
stationsInputFileName	ascii	List of stations		all
nlGmiRestart:				
restart_infile_name	nc	restart file	yes	all
nlGmiEmission:				
emiss_infile_name	nc	emission data	yes	all
fertscal_infile_name	ascii	fertilizer scale data	yes	all
lai_infile_name	ascii	leaf area index data	yes	all
light_infile_name	ascii	light data	no	all
precip_infile_name	ascii	precipitation data	yes	all
soil_infile_name	ascii	soil type data	no	all
veg_infile_name	ascii	vegetation data	yes	all
isopconv_infile_name	ascii	isoprene conversion data	no	all
monotconv_infile_name	ascii	monoterpene conversion data	no	all
laiMEGAN_InfileName	nc	AVHRR leaf area index data	yes	all
aefMboMEGAN_InfileName	nc	methyl butenol annual emiss factors	yes	all
aefIsopMEGAN_InfileName	nc	isoprene annual emiss factors	yes	all
aefOvocMEGAN_InfileName	nc	other VOC annual emiss factors	yes	all
aefMonotMEGAN_InfileName	nc	monoterpene annual emiss factors	yes	all
scFactorNOff_infile_name	nc	Scaling factor for NO fossil fuel emission	yes	T/C
scFactorNObb_infile_name	nc	Scaling factor for biomass burning emission	yes	T/C
emiss_aero_infile_name	nc	aerosol emissions	yes	A/MA/GA
emiss_dust_infile_name	nc	dust emissions	yes	A/MA/GA
GOCARTerod_infile_name	nc		yes	GA
GOCARTocean_infile_name	nc		yes	GA
GOCARTerod_mod_infile_name	nc		yes	GA
nlGmiChemistry:				
AerDust_infile_name	nc	aerosol/dust concentrations	yes	T/C
forc_bc_infile_name	ascii	forcing boundary conditions	no	

loss_data_infile_name	ascii	loss frequency data	no	
h2oclim_infile_name	nc	Water climatology data	yes	
lbssad_infile_name	nc	Liquid binary sulfate	yes	
rxnr_adjust_infile_name	nc	Reaction rate adjustment data	yes	
nlGmiPhotolysis:				
cross_section_file	ascii	X-Section quantum yield	no	
T_O3_climatology_file	ascii	T & O3 climatology	no	
scattering_data_file	ascii	Aerosol/cloud scattering data	no	
rate_file	ascii	master rate data	no	
qj_infile_name	nc	photolysis rates	yes	GA
sfalbedo_infile_name	ascii	surface albedo data	yes	
uvalbedo_infile_name	ascii	uv-albedo data	yes	

Table 4.1: List of namelist variables refering to input file names. The chemical mechanisms are labeled as follow: aerosol (A), gocart_aerosol (GA), micro_aerosol (MA), troposphere (T), and strat_trop (C).

In addition to the files mentioned in Table 4.1, GMI needs meteorological (metFields) input files. The code uses metFields from various sources. The available metFields files are listed in Table 4.2.

Source	Description	Year	Resolution
DAO	GEOS-1 Strat	1997, 1998	$4 \times 5 \times 46$
GISS	GISS prime	1977	$4 \times 5 \times 23$
fvGCM	GMAO GEOS4 AGCM	1994, 1998	$4 \times 5 \times 42$
		1994, 1995, 1996, 1997, 1998	$2 \times 2.5 \times 42$
GEOS4DAS	GMAO GEOS4 DAS	2001, 2004, 2005, 2006, 2007	$4 \times 5 \times 42$
		2000, 2001, 2004, 2005, 2006, 2007	$2 \times 2.5 \times 42$
GEOS4FASM	GMAO GEOS4 Fcst	2001	$2 \times 2.5 \times 42$
GEOS4FCST	GMAO GEOS4 first look Fcst	2004, 2005	$2 \times 2.5 \times 42$

Table 4.2: Available metFields files.

The namelist setting of the metFields files is done through the variable

```
met_infile_names(), or
met_filnam_list
```

4.3 Output Files

ASCII output and binary output files in netCDF data format are produced from GMI runs. The contents and the number of different output files can be controlled by using appropriate namelist parameters. To obtain information on how these parameters are set, please refer to Appendix F.

4.3.1 ASCII Diagnostic Output File

The following rules apply to the ASCII diagnostic output file:

- Is named <problem_name>.asc.
- Contains up to five sections, each of which can be turned on or off through namelist settings.
- For the first three sections, only information on a single specified species is output.
- Can specify a particular longitude index to use in the second section.
- Can specify the output frequency (in number of time steps).

4.3.2 netCDF Output Files

The netCDF output files that can be produced by the GMI model are:

1. <problem_name>.const.nc
   - species concentration
   - [+mass]
   - [+ pressure and/or temperature]
   - [+dry depos. and/or wet depos.]
2. <problem_name>.freq#.nc (# is 1, 2, 3, 4)
3. <problem_name>.overpass#.nc (# is 1, 2, 3, 4)
4. <problem_name>.aerdust.nc
5. <problem_name>.tend.nc
6. <problem_name>.col.nc
7. <problem_name>.cloud.nc
8. <problem_name>.flux.nc
9. <problem_name>.qj.nc
10. <problem_name>.qk.nc
11. <problem_name>.qqi.nc
12. <problem_name>.qqk.nc
13. <problem_name>.sad.nc

The user can

- Output or not any of the above netCDF files
- Specify snapshots or mean values for most of them.
- Specify frequency of output (by number of days, monthly, and/or the 1st and 15th of each month).

In addition to these files, the code can also produce a netCDF restart file named <problem_name>.rst.nc. The file contains everything needed to do a continuation run and can be written out with the frequency (namelist variable *pr_rst_period_days*): number of days, monthly, and/or the 1st and 15th of each month. There is a namelist variable (*do_overwrt_rst*) specifying whether to over-write or append to the file.

In Appendix C, we provide tables listing the contents of the netCDF output files and the frequency that variables in them can be written out.

4.4 Location of Input, Output, and Other GMI Files

The input and output data files are organized by directory on both dirac and discover. The main GMI directory on dirac is **/archive/anon/pub/gmidata2/** and on discover is **/discover/nobackup/projects/gmi/gmidata2**. The directory structure below this level is identical on both machines.

There are 5 major subdirectories underneath **gmidata2**

- **input** (Input files for the GMI model)
- **output** (GMI model output files)
- **docs** (Documentation for the GMI model)
- **progs** (GMI scripts and programs)
- **users** (Files and directories generated and used by GMI users)

4.4.1 Directories under input/

The input files are organized into one of the following 5 subdirectories under **input/**

- **metfields**
- **chemistry**
- **emissions**
- **species**
- **run_info**

The metfields are then further organized by the model from which they were generated (e.g., dao, geos4das, fvgcm), the resolution (e.g., 2x2.5, 4x5), and year (e.g., 1997, 2004) in subdirectories.

The emissions files are ordered according to chemical/aerosol mechanism for which they are used (e.g., combo, trop, strat, gocart, htap). Note that files that are commonly used in many or all GMI runs (containing information about soil type, vegetation type, leaf area index, etc.) are located at this directory level in **common/**.

The input files for GMI chemistry are divided up into **photolysis/**, **surfareadens/**, **aerodust/** and **misc/**. The **misc/** directory contains files that are often used for the runs containing input water, methane, and seasalt data, for example. Under **photolysis/**, the files are further subdivided into **lookup/**, **fastj/**, **fastjx/**, **fastjx53c/**, and **uvalbedo/**.

Under **species/**, there are two subdirectories: **fixed/** and **initial/**. The files under **fixed/** contain data for constituents whose concentrations do not change much with time, such as acetone and methane. The **initial/** directory contains files that can be used to start a model run if a restart file is not available, especially for runs focusing on aerosols. Currently, there is an **aerosol/** directory with relevant input files.

The files under **run_info/** include representative namelists and restarts files that have been used. Also under **run_info/** are potentially useful scripts for running the model and

generating namelist files. The namelists and restarts for particular model runs are generally stored under the output directory for the experiment. See Section 4.4.2 for more details.

Please note that the **input/** directory is updated periodically so that the copy on discover matches that on dirac.

4.4.2 Directories under output/

The directories under **output/** are organized by chemical mechanism:

- gmic (combo model)
- gmia (aerosol model)
- gmit (troposphere model)
- gmis (stratosphere model)

Please note that the output data from the GMI production runs are located only on dirac.

Below each of the above subdirectories, the data are then organized by experiment name (e.g., aura3, aura2for12h, etc.). Each experiment directory is further subdivided by year and under each year directory, the appropriate files are placed in either **stations/**, **diagnostics/**, or **run_info/** directories. If data from the spinup runs used to start the experiment are available, they will be placed in the **spinup/** directory at the level of the year directory.

Chapter 5

Performing Specific Runs

5.1 The GMI Self Contained Test

For testing the model, we provide the directory *gmi_gsfc/SelfContainedTest*. We assume that the test is being run on `discover` and uses the combo (strat_trop) mechanism. In this section we provide intructions for running a 2x2.5 resolution combo model simulation.

The latest version of the code requires an appropriate namelist file. We provide this namelist file, *gmi_gsfc/SelfContainedTest/namelist2x2.5.in*. This namelist file does not require modification to run the test. All input files required to run the test are located in the *gmi_gsfc/SelfContainedTest/input-2x2.5* directory; the namelist uses relative paths to locate the input files here.

Edit the file *gmi_gsfc/SelfContainedTest/run_ref* and provide the full location of the *gmi_gsfc/SelfContainedTest* directory (no relative paths) by setting

```
setenv workDir /discover/nobackup/userid/gmi_gsfc/SelfContainedTest
```

to the appropriate path.

The test can now be run by typing

```
%qsub run_ref
```

Once the test has finished the standard output can be checked for a succesful run. The test run should have created one day's worth of output data. The standard output file should include a statement such as

```
--------  Successful completion of the run  -----------
```

5.2 Standard Runs

The two files mentioned in the preceding section, *gmi_gsfc/SelfContainedTest/namelist2x2.5.in* and *gmi_gsfc/SelfContainedTest/run_ref* can be used a basis for other types of runs, like a troposphere run, for example. To use the various model configurations, the code will need to be compiled with one of the following options:

```
setenv CHEMCASE troposphere
setenv CHEMCASE stratosphere
setenv CHEMCASE strat_trop
setenv CHEMCASE aerosol
setenv CHEMCASE micro_aerosol
setenv CHEMCASE gocart_aerosol
```

The $CHEMCASE$ option should also be changed in the *run_ref* file.

5.3 Other Namelist Variables

Three important namelist variables are used to set the number of processors. They are $numWorkerProcs$, $numLonProcs$ and $numLatProcs$, and they satisfy the relation

$$numWorkerProcs = numLonProcs \times numLatProcs.$$

To execute the code, the number of processors $Ncpus$ is given by $Ncpus = numWorkerProcs+1$.

The resolution of the run is set using the following variables: $i1\_gl$ and $i2\_gl$ (first and last index of the global longitude); $ju1\_gl$ and $jv1\_gl$ (first global 'u' and 'v' latitude); $j2\_gl$ (last global 'u&v' latitude); kl_gl and $k2\_gl$ (first and last global altitude).

Remark 3 *The resolution of the run that is specified by the above variables should match the resolution of the metfield files. Metfield files can be found on* `discover` *in*

```
/discover/nobackup/projects/gmi/gmidata2/input/metfields/TYPE/RESOLUTION/
```

and on explore/palm in

```
/explore/nobackup/projects/gmi/gmidata2/input/metfields/TYPE/RESOLUTION/
```

The metfield files are listed in a file specified by the namelist variable met_filnam_list.

5.4 Restart Capabilities

To restart from a previous run, in the nlGmiRestart section of the input namelist file, set

```
pr_restart         = T,
pr_rst_period_days = #.#d0,
```

Replace "#.#" with whatever you want the time period to be.
Note that pr_rst_period_days when converted to seconds, must be a multiple of mdt, the time increment for reading in new met data. Restart netCDF output will be written to a file named

```
<problem_name>.rst.nc
```

Set the namelist variable, *do_overwrt_rst* as appropriate (i.e., do you want to keep just a single record of restart data or multiple records?).

To create a new namelist input file to use for running from a restart point:

1. Section 7 provides information on a script called *MakeNameLists.py* that can create multiple namelist files and handle restarting.

2. Alternatively, you can execute the *mk_rstnl* ("make restart namelist") script as follows

```
$gmi/Shared/GmiScripts/mk_rstnl \
   -onl <old_nlfile>   -nnl <new_nlfile>  \
   -rst <rst_ncfile>   -eda <endGmiDate>  -eti <endGmiTime> \
  [-npn <problem_name>]

The first four arguments/value pairs are required

  old_nlfile   : name of the namelist input file that was used to get to
                 the restart point
  new_nlfile   : name to call the new namelist input file that will be
                 used to continue from the restart point
  rst_ncfile   : name of the netCDF restart file that was written out at
                 the restart point
  endGmiDate   : ending date for the new run; note that begGmiDate will
                 be set by the script to what endGmiDate was when the
                 netCDF restart file was written out
  endGmiTime   : ending time for the new run; note that begGmiTime will
                 be set by the script to what endGmiTime was when the
                 netCDF restart file was written out

The last argument/value pair is optional

  problem_name : new problem name to use in the new namelist input file
```

For example

```
$gmi/Shared/GmiScripts/mk_rstnl -onl nlfile.in.old  -nnl nlfile.in.new \
  -rst ncfile.rst.nc  -eda 20061231 -eti 240000
```

This script automatically creates a new namelist file with various namelist variables modified or added to accomodate the restart. The script uses information from the netCDF restart file to accomplish this. Values that the script will use are output to the screen; these should be reviewed to verify that they are what you expected them to be.

The namelist variables that the script will potentially modify or add are:

```
nlGmiControl =>
  problem_name        : set to value provided by -npn script argument;
                        otherwise unchanged from value in old_nlfile
  begGmiDate          : set to endGmiDate at restart point
  begGmiTime          : set to endGmiTime at restart point
  endGmiDate          : set to value provided by -tfi script argument
  endGmiTime          : set to value provided by -tfi script argument
  gmi_sec             : set to ending value at restart point

nlGmiMetFields   =>
  met_infile_num      : set to ending value at restart point
  mrnum_in            : set to ending value at restart point - 1
  tmet1               : set to tmet2        at restart point

nlGmiDiagnostics  =>
  pr_qqjk             : set to ending value at restart point

nlGmiRestart =>
  rd_restart          : set to T
  restart_infile_name : set to value provided by -rst script argument
```

Note that restart dumps can only occur at the end of a met data cycle and you should edit the new namelist input file directly if you want to change any other namelist variables not listed above.

If there is more than one record in the restart file, the default is to use the last one. This can be changed by setting the namelist variable, *restart_inrec*, to a different record number.

To resume running from the restart point:

- Start the run just as you normally would, but this time use the new restart namelist input file you created above.

Chapter 6

Making Changes to the Code

This chapter briefly describes some factors users should take into account if they want to modify the code.

6.1 Coding Consideration

Some coding conventions must be followed when making code changes:

- All real variables must be declared "real*8".
- All real numbers must have a "d" exponent even if "d0".
- Must use "# include" for all include statements.
- Use only generic intrinsic function calls (for instance use MOD instead of AMOD).
- Do not use the real or float intrinsic functions: just assign integer variable to a real*8 variable if need be and then proceed.
- Avoid using machine-specific calls.
- Use of appropriate F90 language features is encouraged (dynamic allocation, array syntax, etc.).
- Use the physical constants and conversion factors defined in Shared/GmiInclude/gmi_phys_constants.h.
- Use the time constants and conversion factors defined in Shared/GmiInclude/gmi_time_constants.h.

6.2 Making Changes in the Code

Before making any change in the code, it is important to understand its directory structure and the concept of components. The code has been reorganized and major components were isolated with the goals of making the code more readable, flexible, accessible, modular

and easy to maintain. We grouped variables belonging to a given component into a derived type. The following derived types were created to classify and manipulate GMI variables:

Advection: Contains advection related variables

Chemistry: Contains chemistry related variables

Convection: Contains convection related variables

Deposition: Contains deposition related variables

Diffusion: Contains diffusion related variables

Emission: Contains emission related variables

SpeciesConcentration: Responsible for manipulating species concentration related variables.

For each member variable (kept private and only directly accessible by the component it belongs to) of the derived type, we wrote routines (whenever necessary) to manipulate it: *Allocate* (to allocate the variable once), *Set* (to set the value of the variable anywhere in the code), and *Get* (to obtain the value of the variable anywhere in the code). It is important to note that the three routines are the only operations directly done on derived type member variables.

For each component, we wrote three new interface routines to standardize the way components are handled:

InitializeComponent: called once to read namelist file (only the section corresponding to the component), do initial settings and perform variable allocation.

RunComponent: used to invoke the component for model time stepping.

FinalizeComponent: called if necessary to deallocate variables.

The above routines are the only ones accessible by the main program and they completely hide the legacy system. They have as arguments only derived types. The only way to have access to a member variable of a derived type is through the associated *Set* and *Get* routines.

In addition to the components listed above, we created three other components to provide services to the seven main components. They are:

gmiClock: Contains clock information necessary for the model to advance in time.

gmiGrid: Contains grid-related data.

gmiDomain: Contains subdomain information.

MetFields: Responsible for updating all the meteorological related variables (by reading from a file or deriving from existing variables), and passing them to other components as needed.

Diagnostics: Responsible for setting the necessary diagnostics flags and passing the appropriate information to other components (that will do proper allocation of diagnostics variables they own).

Example 1 *We want to provide an example on how the componentization was used to drive the Chemistry component. We list the arguments of the three routine interface:*

```
InitializeChemistry(Chemistry, Diagnostics, namelistFile, gmiGrid, gmiDomain)
RunChemistry       (Chemistry, Emission, MetFields, Diagnostics,
                    SpeciesConcentration, gmiClock, gmiGrid, gmiDomain)
FinalizeChemistry  (Chemistry)
```

Here, `namelistFile` *is the namelist file.*

The code associated with the first set of derived types is located in the *Components/* directory whereas the one for the second set is in the *Shared/* directory (providing support to major components). When making changes in the code, we need to identify where the modifications should occur. Assume that we want to add a new method for computing the tropopause pressure. The new routine should be added in the directory *Shared/GmiMetFields/* containing modules/routines manipulating metFields variables.

6.2.1 Adding a Variable to a Derived Type

Each derived type defined in the code is part of a module. To add a new variable to a derived type, we need to do the following operations:

1. Define the variable
2. Write a routine to allocate it (if an array)
3. Write a routine to get its value (if needed outside the component the derived type belongs to)
4. Write a routine to set its value (if updated outside the component the derived type belongs to)
5. Deallocate the derive type (if an array)

As an example, assume that we want to output the overhead ozone column that is calculated inside the photolysis package. The variable `overheadO3col` needs to be created. Since it will only be updated in the Chemistry component (where the Photolysis is located), the variable will be part of the Chemistry component and should be a member of the Chemistry derived type. In addition, `overheadO3col` will be visible outside the component. The following will be added in the file:

```
gmi_gsfc/Components/GmiChemistry/chemistryMethod/GmiChemistryMethod_mod.F90
```

```
real*8, pointer     :: overheadO3col(:,:,:)   => null()

subroutine Allocate_overheadO3col (self, i1, i2, ju1, j2, k1, k2)
  integer           , intent(in   )  :: i1, i2, ju1, j2, k1, k2
  type (t_Chemistry), intent(inout)  :: self
  Allocate(self%overheadO3col(i1:i2, ju1:j2, k1:k2))
  self%overheadO3col = 0.0d0
  return
end subroutine Allocate_overheadO3col

subroutine Set_overheadO3col (self, overheadO3col)
  real*8          , intent(in)  :: overheadO3col(:,:,:)
  type (t_Chemistry), intent(inout)   :: self
  self%overheadO3col(:,:,:) = overheadO3col(:,:,:)
  return
end subroutine Set_overheadO3col

subroutine Get_overheadO3col (self, overheadO3col)
  real*8          , intent(out)  :: overheadO3col(:,:,:)
  type (t_Chemistry), intent(in)   :: self
  overheadO3col(:,:,:) = self%overheadO3col(:,:,:)
  return
end subroutine Get_overheadO3col
```

6.2.2 Adding Chemical Mechanisms

Currently, the GMI code has six chemical mechanisms:

1. *aerosol*
2. *micro_aerosol*
3. *gocart_aerosol*
4. *stratosphere*
5. *strat_trop* (combined stratosphere/troposphere)
6. *troposphere*

The code portion for each mechanism is contained in its own subdirectory (located at *Components/GmiChemistry/mechanisms/*) having the name of the corresponding mechanism. It is organized into two subdirectories (see Chapter 2): *include_setkin/* and *setkin/*.

If you want to add another chemical mechanism, just create a new subdirectory from *Components/GmiChemistry/mechanisms/* and move the setkin files there. To compile the code, follow the compilation procedures as described in Section 3.4.

The selection of a particular chemical mechanism is done through the environment variable *CHEMCASE* in the file *cshrc.gmi* (see Chapter 3).

Remark 4 *It is important to note that if you make changes specific to a particular chemical mechanism (i.e., outside setkin files), use the variable "chem_mecha" to delimit your changes.*

6.2.3 netCDF Output Files

We modified the process of producing netCDF output files. All the operations needed to manipulate a file are now included in a unique Fortran module. In addition, the interface routines (initialize, control and finalize) have standard interfaces and should not be changed. If we want to add another diagnostics variable to a netCDF output file, we may only have to make changes in the module: declare the variable, allocate it, update its value every time step, communicate its value to the root processor, write it out by the root processor and deallocate it.

Chapter 7

Script Tools

7.1 Internal Scripts

The GMI code comes with several scripts that allow users to perform commands such as searching from words within the code, counting the number of lines in the source code, constructing a restart input namelist file, etc. All the scripts are located in the directory *Shared/GmiScripts* and can be executed from anywhere in the code. They are:

check_ver searches for all file names in the current directory containing "search_string" and replaces the first instance of "search_string" with "replace_string".
Usage: chname search_string replace_string

clgem deletes a number of the files created when *gmi* is run.

doflint runs the flint Fortran source code analyzer on the GMI code.
It can be run on any machine where flint is available (e.g., tckk), and where the GMI code has been installed and compiled.
Usage: doflint

gmi_fcheck can be used to run the Flint source code analyzer on gmi_gsfc.

gmi_fcheckrm can be used after "gmi_fcheck" is run on the gmi_gsfc code to strip out Flint messages that are of no consequence. Gmi_fcheckrm uses gmi_fcheck.out as its input file, and produces a new file called gmi_fcheckrm.out.

grabf grabs all of the ".f90" files and puts them in $gmi/CODE. These files then can be ftped to a machine with access to the FORTRAN Lint (flint) source code analyzer tool.

lastmod lists all files in reverse order of when they were last modified. It is useful for determining which routines a user has modified since they were last installed in the code.
Usage: lastmod [tail_num]

lastmod_all lists all files in reverse order of when they were last modified. It is useful for determining which routines a user has modified since they were last installed in the

code.
Usage: lastmod_all [tail_num]

line_count_gmi does a variety of line counts on the GMI source code.

list_species lists all the species labels for the selected chemical mechanism (environment variable CHEMCASE).
Usage: list_species

lnsdat (lns (symbolic link) data) symbolically links the GMI input file directory to "gmi_data" in the current directory. The GMI namelist file can then point to a generic gmi_data directory.

mk_rstnl (make restart namelist) constructs a restart input namelist file (see Section 5.4 for more information).

savit creates a clean copy of a GMI code tree (considered only files with extension [.F90—.c—.h]) in a tmp directory, tars it up into a tarfile named gmisav.tar, and puts this file in the directory where your gmi directory resides. The tmp files are then deleted.

savset creates a clean copy of the GMI setkin files (considered only files with extension [.F90—.c—.h]) in a tmp directory, tars it up into a tarfile named setsav.tar, and puts this file in the directory where your gmi directory resides. The tmp files are then deleted.

seabf searches through the ".F90" source files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.
Usage: seabf search_string [file_name]

seabf_word searches through the ".F90" source files for a particular word (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.
Usage: seabf_word search_string [file_name]

seac searches through the .c source files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.
Usage: seac search_string [file_name]

seah searches through the ".h" include files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.
Usage: seah search_string [file_name]

seah_word searches through the ".h" include files for a particular word (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.
Usage: seah_word search_string [file_name]

sealf searches through the ".f90" source files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.
Usage: sealf search_string [file_name]

seamf searches through the "Makefile" and "Makefile.cpp" source files for a particular string (case insensitive). Output comes to the screen unless an optional second file_name argument is provided.
Usage: seamf search_string [file_name]

7.2 Production Tools

There is a collection of python scripts for doing GMI production runs. These scripts create namelist files for each segment of the production run (usually date segments), start each segment of a production run, monitor the runs, and sends status updates to a designated email address. The following scripts can be used seperately or together for doing productions runs:

MakeNameLists.py creates namelist files by using a base namelist

ChainRuns.py runs multiple segments of a GMI production run, monitors the progress, and sends status updates

The first step in using either of these scripts is to edit the *ExperimentConstants.py* file. The following are the configurable options inside this file:

MAILTO a single email address or a comma seperated list of email addresses to send updates to

NUM_CPUS total number of processors (slaves+master)

BASE_NAMELIST the base namelist file

NUM_NAMELISTS the number of total namelists to be create. We suggest setting this to 12

START_MONTH usually "jan" for january; all other months follow the same three letter abbreviation

NUM_ARGS only developers should change this option

NAMELISTS the name of the file the will contain a list of all the namelists in the production run; they are listed in the order of execution

RUNREF the base batch file for submission to the discover PBS batch system

QUEUEMINUTESALLOWED the number of minutes a segment should take to run, including the estimated wait time in the queue

Next, a run directory should be created. The run directory should have a sample RUNREF file and a BASE_NAMELIST in it; in this example we use the files *run_ref* and *sample.in*, respectively.

In the *run_ref* file, make sure you change the *group_list*, the *workDir* (which is the run directory), and the path to the executable (GEMHOME). Also, check the number of cpus.

For the BASE_NAMELIST file, check that the restart file is correct, and other input file names (anything that contains a string *infile_name*). This file will be used as a base for all your production segments. The job name, date, restart file (previous month), and the number of days to run will be changed by the script.

Remark 5 *If you limit the number of segments to 12 (one year) you should not have to make any other modifications EXCEPT when you are intending to run a February segment with 29 days. You will have to change the namelist variable (after you run the* MakeNameLists.py *script) to tfinal_days to 29. Users are urged to use caution when using the* MakeNameLists.py *and check their work carefully.*

Once you've prepared the run directory execute the *MakeNameLists.py* script from the run directory (substitute your own directory names here):

```
% cd runDirectory
% python /yourpath/MakeNameLists.py -d ‘pwd‘
```

You should see output similiar to this:

```
nameListPath = /yourpath
nameListFile = sample.in
numberOfNameLists = 12
startMonth = jan
startYear = 1994
start month 0 jan
gmit_sample_1994_feb feb1994.list 28 940201 940228 gmit_sample_1994_jan
gmit_sample_1994_mar mar1994.list 31 940301 940331 gmit_sample_1994_feb
gmit_sample_1994_apr apr1994.list 30 940401 940430 gmit_sample_1994_mar
gmit_sample_1994_may may1994.list 31 940501 940531 gmit_sample_1994_apr
gmit_sample_1994_jun jun1994.list 30 940601 940630 gmit_sample_1994_may
gmit_sample_1994_jul jul1994.list 31 940701 940731 gmit_sample_1994_jun
gmit_sample_1994_aug aug1994.list 31 940801 940831 gmit_sample_1994_jul
gmit_sample_1994_sep sep1994.list 30 940901 940930 gmit_sample_1994_aug
gmit_sample_1994_oct oct1994.list 31 941001 941031 gmit_sample_1994_sep
gmit_sample_1994_nov nov1994.list 30 941101 941130 gmit_sample_1994_oct
gmit_sample_1994_dec dec1994.list 31 941201 941231 gmit_sample_1994_nov
```

You may need to make a change to the *sample.in* namelist. Open the *namelists.list* file and change the name of the first namelist to be consistent with the rest. For example: (you can avoid this step by naming the BASE_NAMELIST something like gmit_sample_1994_jan.in)

Before:

```
sample.in
gmit_sample_1994_feb.in
gmit_sample_1994_mar.in
gmit_sample_1994_apr.in
gmit_sample_1994_may.in
gmit_sample_1994_jun.in
gmit_sample_1994_jul.in
gmit_sample_1994_aug.in
gmit_sample_1994_sep.in
gmit_sample_1994_oct.in
gmit_sample_1994_nov.in
gmit_sample_1994_dec.in
```

After:

```
gmit_sample_1994_jan.in
gmit_sample_1994_feb.in
gmit_sample_1994_mar.in
gmit_sample_1994_apr.in
gmit_sample_1994_may.in
gmit_sample_1994_jun.in
gmit_sample_1994_jul.in
gmit_sample_1994_aug.in
gmit_sample_1994_sep.in
gmit_sample_1994_oct.in
gmit_sample_1994_nov.in
gmit_sample_1994_dec.in
```

Rename the file that you just changed. For example:

```
%mv sample.in gmit_sample_1994_jan.in
```

Remark 6 *Make sure you have ALL your metFields files in the run directory, as the* ChainRuns.py *script assumes this location.*

Modify your crontab to run the *ChainRuns.py* script at the proper time. For additional documentation on the crontab tool please reference:

```
http://www.adminschoice.com/docs/crontab.htm
```

Here is an example of a ChainRuns crontab entry:

```
20 11 15 6 * /usr/bin/python /yourPath/ChainRuns/ChainRuns.py -d runDirectory >>
```

In this example, the *ChainRuns.py* script will run at 11:20 am on June 15th.

Chapter 8

How to Use CVS

8.1 What is CVS?

CVS is an acronym for the "Concurrent Versions System". It is a "Source Control" or "Revision Control" tool having the following features:

- Non-proprietory and can be downloaded from the internet;
- Allows users to work simultaneously on the same file, keep track of changes by revision, tag and date;
- Can obtain an earlier version of the software easily;
- Allows the user to track the supplier's software releases while making code changes locally.
- Enables the user to merge code changes between his version and supplier's automatically and identify problems if merge presents contradictions;
- A user of CVS needs only to know a few basic commands to use the tool.

Here are some important terms used with CVS:

Repository: The directory storing the master copies of the files. The main or master repository is a tree of directories.

Module: A specific directory (or mini-tree of directories) in the main repository. Modules are defined in the CVS modules file.

RCS: Revision Control System. A lower-level set of utilities on which CVS is layered.

Check out: To make a copy of a file from its repository that can be worked on or examined.

Revision: A numerical or alpha-numerical tag identifying the version of a file.

8.2 How to Use CVS

There are two ways you can use CVS:

1. Use CVS to keep up-to-date with the GMI code changes. This will require a *sourcemotel* account.

2. Use CVS to track both GMI code releases and your own changes. You can do this either on *sourcemotel* or on your local machine (with your own CVS installation).

8.3 Use CVS to Keep Up-to-Date with GMI Source Code Changes

CVS is used to keep track of collections of files in a shared directory called "The Repository". Each collection of files can be given a "module" name, which is used to "checkout" that collection. After checkout, files can be modified (using your favorite editor), "committed" back into the Repository and compared against earlier revisions. Collections of files can be "tagged" with a symbolic name for later retrieval. You can add new files, remove files you no longer want, ask for information about sets of files in three different ways, produce patch "diffs" from a base revision and merge the committed changes of other developers into your working files. In this section, we explain how these operations are done with the GMI code. It is assumed that the user has an account on *sourcemotel* and that CVS is installed on his local computer.

We assume that you have already obtained a copy of the code (say the most recent release) from *sourcemotel* by using the command:

```
%cvs -d usrid@sourcemotel.gsfc.nasa.gov:/cvsroot/gmi co gmi_gsfc
```

A directory labeled *gmi_gsfc* (containing the code) will be created at the location where the command was executed.

Assume that you want to know all the different available releases (with the associated tags) of the GMI code. From the *gmi_gsfc* directory, type

```
%cvs status -v Makefile
```

to obtain the status of the file *Makefile*. The results give (first few lines):

```
===================================================================
File: Makefile          Status: Up-to-date

   Working revision:    1.18
   Repository revision: 1.18    /cvsroot/gmi/gmi_gsfc/Makefile,v
   Sticky Tag:          (none)
   Sticky Date:         (none)
   Sticky Options:      (none)

   Existing Tags:
```

```
fromGEMHOME_to_GMIHOME          (revision: 1.18)
NOxEmissionsScalingFactors_v3   (revision: 1.17)
NOxEmissionsScalingFactors_v2   (revision: 1.17)
Lightning_Branch_fvgcm_ap1_0    (revision: 1.14)
NOxEmissionsScalingFactors_v1   (revision: 1.17)
Lightning_Branch_fvgcm_default  (revision: 1.14)
Lightning_Branch_DAS_default_fix        (revision: 1.14)
Lightning_Branch_DAS_default    (revision: 1.14)
Lightning_Branch_DAS_ap1_0      (revision: 1.14)
SurfaceConstituents_for_ColumnDiagnostics       (revision: 1.17)
ImplementationMEGANv1           (revision: 1.17)
Flux_Freq_Routines_in_Modules   (revision: 1.16)
GeorgiaTechCloudModule_v2       (revision: 1.16)
HorizontalDomainForFreqOutputs  (revision: 1.16)
GeorgiaTechCloudModule_v1       (revision: 1.16)
```

One can observe that the code has four revisions (1.18, 1.17, 1.6 and 1.14) in the above tags.

```
%cvs export [-D today][-r tag] gmi_gsfc
```

gives an exported version of the *gmi_gsfc* directory. The expressions in [ ] are options. '-D today' gives the latest version of the code. The user can also specify "-D 'September 26, 2007'" (note that the date is in single quotes) for the version from that day, or use '-r release-1-17' for release 1.17 (release-1-17 is a CVS tag), or '-r NOxEmissionsScalingFactors_v3' for the *gmi_gsfc* directory with tag NOxEmissionsScalingFactors_v3.

```
%cvs checkout gmi_gsfc
```

provides in addition to exported version, CVS information. With such information, users will be able to keep up-to-date with our release automatically with the simple `cvs update` command (instead of having to manually insert the changes we broadcast). Once you check out a version of the code, you form a 'working directory'.

```
%cvs update gmi_gsfc
```

only works if a user has a cvs-checked-out version. This brings the changes made in the master repository to the user's working directory. The user may only want to know which files were modified without making any update:

```
%cvs -n update gmi_gsfc
```

An example of the print out from the cvs update command:

Example 2 *You want to checkout a copy of the GMI code from* sourcemotel. *Then type*

```
%cvs checkout gmi_gsfc
```

It creates a copy of the code in your own directory. Assume someone else has made some changes in the code and the next code release is available. You can simply do a `cvs update` *to bring the new changes in the new release into your copy:*

```
%cd gmi_gsfc
%cvs update
```

A list is printed on your screen to let you know which files were updated (a 'U' in front of the file) from the new release, and which files were modified (a 'M' in front of the file) and any conflict that may result from this update.

Remark 7 *Note that doing* **cvs update** *under the* gmi_gsfc *directory will automatically update the entire code. You can update an individual directory or file by going into the directory and issuing* **cvs update**- *which updates that directory and any sub-directories, or* **cvs update filename**- *which updates only that file.*

```
%cvs diff filename
```

This does the differencing between the file in your working repository with the one you checked out from the *sourcemotel* repository.

Example 3 *Assume that you want to compare the file* Makefile *(inside gmi_gsfc) from your working repository with the one on* sourcemotel *in the release with TAG* NOxEmissionsScalingFactors_v3*:*

```
%cvs diff -r NOxEmissionsScalingFactors_v3 Makefile

Index: Makefile
===================================================================
RCS file: /cvsroot/gmi/gmi_gsfc/Makefile,v
retrieving revision 1.17
retrieving revision 1.18
diff -r1.17 -r1.18
9,11c9
< LIBS = -L$(LIB_DIR)
<
< #FFLAGS+=$(INCS)
---
> LIBS = -L$(LIB_DIR)
38,39c36,37
< #all: packageddir shared components
< all: packageddir shared components applications
---
> new: packageddir shared components applications
> all: packageddir shared components applications legacy
48a47,49
> legacy:
```

```
>       (cd $(GMIHOME)/gem; $(MKMF); make)
>
57,58c58,59
< #     @$(MAKE) -C $(APPLICATIONS) EmissionDriver.ex
< #     @$(MAKE) -C $(APPLICATIONS) DiffusionDriver.ex
---
> #       @$(MAKE) -C $(APPLICATIONS) EmissionDriver.ex
> #       @$(MAKE) -C $(APPLICATIONS) DiffusionDriver.ex
68a70
>       (cd $(GMIHOME)/gem; make clean)
```

To list log messages and status of the master repository, issue the command:

```
%cvs log filename
```

8.4 Use CVS to Track Both New Releases and Your Changes

If you want to maintain your own code and keep track of the changes from *sourcemotel*, what you should do is create your own repository and use the 'vendor branch' concept in CVS. If you do it from your local machine, set

```
setenv CVSROOT some-home-directory-on-your-local-machine
```

(e.g. setenv CVSROOT /home/userID/gmi_ repository)
in your *.cshrc* file.
To initialize the repository, type

```
%cd /home/userid/gmi_repository
%cvs init
```

Now you can checkout any release of the GMI code. Assume that you want to obtain the release *HorizontalDomainForFreqOutputs*

```
%cvs -d usrid@sourcemotel.gsfc.nasa.gov:/cvsroot/gmi co -r  \
     HorizontalDomainForFreqOutputs gmi_gsfc
```

If you only want the *Components/* directory, type

```
%cvs -d usrid@sourcemotel.gsfc.nasa.gov:/cvsroot/gmi co -d Components -r  \
     HorizontalDomainForFreqOutputs gmi_gsfc/Components
```

You can now work with the code. If you make some changes and want to bring them to your repository to keep, do the following from the directory *gmi_repository/gmi_gsfc*:

```
%cvs diff > output.diff
%cvs update
%cvs commit -m 'message for the commit'
```

If you want to create a new file that does not exist in the repository and you want to add it in the repository, type (from the directory where the new file resides)

```
%cvs add new_file
```

Example 4 *Assume that you want to add a new chemical mechanism,* `new_chem`*. In the directory* Components/GmiChemistry/mechanisms*, you have created the directory* `new_chem/` *that contains the subdirectories* include_setkin/ *and* setkin/*. To add the directory structure of the new chemical mechanism into the repository, do the following:*

```
%cd Components/GmiChemistry/mechanisms
%cvs add new_chem/
%cvs commit new_chem/
%cd new_chem
%cvs add include_setkin/
%cvs commit include_setkin/
%cd include_setkin
%cvs add *
%cd ../
%cvs add setkin/
%cvs commit setkin/
%cd setkin
%cvs add *
```

8.5 Where To Obtain CVS

```
https://ccvs.cvshome.org/servlets/ProjectDocumentList
```

Chapter 9

Parallel Performance

Over the past few years, the GMI code has evolved to become a componentized software package. The code has been subject to many modifications in terms of software design and implementation. These modications may add overheads in the computing time and may lead to the deterioration of the parallel performance of the code. Therefore it is important to continually monitor the performance of GMI by profiling the code and by studying how the code scales across processor.

Profiling a code can be defined as the use of software tools to measure a program's run-time characteristics and resource utilization. It is important to identify where the bottlenecks are and why these areas might be causing problems. By utilizing profiling tools and techniques, we want to learn which areas of the code offer the greatest potential performance increase. We want to target the most time consuming and frequently executed portions of the program for optimization with the objective of reducing the overall wall clock execution time.

In this chapter, we only provide a simple analysis to verify if the current version of the code produces similar performance with respect to previous versions. The model integration was carried out on *discover*:

1. Linux Networx Cluster
2. 2560 CPUs
3. Four processors per node
4. 160 GB disk, 4 GB of RAM per node
5. 3.2 GHz of processor speed

We use the combined stratosphere/troposphere chemical mechanism:

- 124 species
- 121 chemical species
- 117 active chemical species
- 68 advected species

- 2×2.5 honrizontal resolution and 42 vertical levels
- model time step of 30 minutes
- one-day integration

We ran the code using 63 and 121 worker processors respectively and recorded the timing information (obtained by setting the namelist variable *do_ftiming*):

```
9x7 = 63 worker processors
   ------------------------------------------------------------------
      Block                        Min Time    Max Time    Avg Time
   ------------------------------------------------------------------
   whole_GMI                      842.9362    843.0141    842.9601
   ------------------------------------------------------------------
   gmiTimeStepping                773.3378    775.4813    774.0801
      procSyncBegStepping           1.0650      3.2106      1.8059
      gmiEmission                   7.5113      8.0610      7.5410
      gmiDiffusion                  2.0009      2.3482      2.1170
      procSyncBeforeAdvection       0.2282      1.1347      1.0033
      gmiAdvection                 77.9177     78.9366     78.8842
      gmiConvection                 4.6655     12.6235      8.4833
      gmiDryDeposition              0.5783      1.1403      0.7806
      gmiWetDeposition              7.1136      8.9838      7.9951
      gmiChemistry                140.5785    497.1423    341.6406
         gmiPhotolysis              5.1924     48.1437     29.2000
      procSyncEndStepping         169.4039    523.4902    322.9148
   gmiWritingOutput                35.1658     45.5605     44.0967

11x11 = 121 worker processors
   ------------------------------------------------------------------
      Block                        Min Time    Max Time    Avg Time
   ------------------------------------------------------------------
   whole_GMI                      541.0739    541.1197    541.1028
   ------------------------------------------------------------------
   gmiTimeStepping                461.5172    467.9213    464.6045
      procSyncBegStepping           0.9933      7.4049      4.0861
      gmiEmission                  11.0396     12.0051     11.2219
      gmiDiffusion                  0.7928      1.3231      1.0657
      procSyncBeforeAdvection       0.6895      2.1456      1.7608
      gmiAdvection                 55.3105     56.1626     55.8819
      gmiConvection                 1.6163      6.0775      4.1244
      gmiDryDeposition              0.2393      0.4921      0.3519
      gmiWetDeposition              3.7472      5.0127      4.3225
      gmiChemistry                 40.9454    322.2511    175.7995
         gmiPhotolysis              0.4408     28.4638     15.5451
      procSyncEndStepping          57.8476    343.3024    205.5430
```

```
gmiWritingOutput                29.5422     35.9619     34.0993
```

Note the following remarks:

1. The profiling is done on the worker processors only and does not included the intialization stage.
2. The presented timing numbers mainly included the profiling of the time stepping routine, gmiTimeStepping:
   - At the beginning of the routine there is an MPI barrier call (procSyncBegStepping)
   - At the end of the routine there is an MPI barrier call (procSyncEndStepping)
3. The Chemistry component is called at the end of gmiTimeStepping. If we focus on gmiChemistry and procSyncEndStepping, we observe a load inbalance within Chemistry. It is more likely due to the solver where the tasks assigned to each processor are not evenly divided.
4. Note that there are only 68 advected species while 117 active chemical species are used. In a previous study (done in 2004), it was observed that if Advection and Chemistry have the same number of species, Advection will dominate calculations as the number of processors increases. It was recommended then to reduce the number of advected species. It is no surprise that Advection is less computational intensive than Chemistry.
5. The computing time decreases as the number of processors increases with a gain of about 38% in time.
6. The results shown here are consistent with what was obtained using previous versions of the code.

Bibliography

[1] D.B. Considine, P.S. Connell, D.J. Bergmann, D.A. Rotman, and S.E. Strahan. Sensitivity of global modeling initiative ctm predictions of anatatic ozone recovery to gcm and das generated meteorological fields. preprint.

[2] D.B. Considine, A.R. Douglass, P.S. Connell, D.E. Kinnison, and D.A. Rotman. A polar stratospheric cloud parameterization for the global modeling initiative three dimensional model and its response to stratospheric aircraft. *J. Geo. Res.*, 105(D3):3955–3973, 2000.

[3] A.R. Douglass, M.J. Prather, T.M. Hall, S.E. Strahan, P.J. Rasch, L.C. Sparling, L. Coy, and J.M. Rodriguez. Choosing meteorological input for the Global Modeling Initiative assessment of high-speed aircraft. *J. Geo. Res.*, 104(D22):27545–27564, 1999.

[4] D.E. Kinnison, P.S. Connell, J.M. Rodriguez, D.A. Rotman, D.B. Considine, J. Tannahill, R. Ramaroson, P.J. Rasch, A.R. Douglass, S.L. Baughcum, L. Coy, D.W. Waugh, S.R. Kawa, and M.J. Prather. The Global Modeling Initiative assessment model: application to high speed civil transport perturbation. *J. Geo. Res.*, 106(D2):1693–1711, 2001.

[5] N. Meskhidze, R.E.P. Sotiropoulou, A. Nenes, J. Kouatchou, B. Das, and J. M. Rodriguez. Aerosol-cloud interactions in the NASA GMI: model development and indirect forcing assessments. *Atmos. Chem. Phys. Discuss.*, 7:14295–14330, 2007.

[6] D.A. Rotman, J.R. Tannahill, D.E. Kinnison, P.S. Connell, D. Bergmann, D. Proctor, J.M. Rodriguez, S.J. Lin, R.B. Rood, M.J. Prather, P.J. Rasch, D.B. Considine, R. Ramaroson, and S.R. Kawa. The Global Modeling Initiative assessment model: model description, integration, and testing of the transport. *J. Geo. Res.*, 106(D2):1669–1691, 2001.

[7] S.E. Strahan, B.N. Duncan, and P. Hoor. Observationally-derived diagnostics of transport in the lowermost stratosphere and their application to the GMI chemistry transport model. *Atmos. Chem. Phys.*, 7:2435–2445, 2007.

Appendix A

Include Files

The files presented here are located in *Config/* directory. They are required for the preprocessing and compilation of the code.

gem_config.h ,*compiler.mk* and *libraries.mk*
Sets configuration parameters for gem Makefiles. The following information must be provided:

- Location of the netCDF include files (variable *INCLUDES_NETCDF*) and library (*NETCDFLibs*)
- Location of the ESMF include files (variable *INCLUDES_Esmf*) and library (*EsmfLibs*)
- Location of the MPI include files (variable *INCLUDES_MSG*) and library (*MSGLIBDIR*)
- Commands to call the C (variable *CC*) and Fortran (*FC*) compilers.
- Compilations options.

gem_options.h
To select the package, the chemical solver, and the desire compilation mode (debugging, optimization, profiling).

gem_sys_options.h
To select the architecture and the message passing options. The user must set the variables:

- *ARCH_OPTION*: to determine the platform used.
- *MSG_OPTION*: for the message passing option. Choose *MSG_NONE* if you want the single processor version of the code, or *MSG_MPI* if you prefer the multiple processor one with MPI as message passing.

- *MPI_2_OPTION*: to determine if you want to include or not MPI-2 calls. Consider *NO_MPI_2* if the platform you will be running the code on does not support MPI-2. Otherwise choose *WITH_MPI_2*.

gem_msg_numbers.h
Contains the message numbers used to identify specific messages in the message passing calls. Should not be edited.

gem_rules.h and *rules.mk*
Contains suffix dependencies and compilation rules for all Makefile/Makefile.cpp files in directories that contain source files. Should not be edited.

With time, we will make necessary changes to remove all the ".h" files and only keep the ".mk" ones.

Appendix B

Single/Multiple Processor Runs

We describe how to carry out single (MPI is not used) or multiple processor runs. Before you compile the code, you need to edit the file *include/gem_sys_options.h* and set

```
#define MSG_OPTION MSG_NONE
```

if you want to produce the executable for a single processor run, or

```
#define MSG_OPTION MSG_MPI
```

for the multiple processor run.

After you obtained the executable, you need to edit your input namelist file.

Namelist Variables for Single CPU

```
oneProcRun     = T
numWorkerProcs = 1
numLonProcs    = 1
numLatProcs    = 1
```

Namelist Variables for Multiple CPUs

```
oneProcRun     = F
numWorkerProcs = 15
numLonProcs    = 5
numLatProcs    = 3
```

The following relationships need to be satisfied in order to run the code:

$$numWorkerProcs = numLonProcs \times numLatProcs$$

$$\frac{i2\_gl - i1\_gl + 1}{numLonProcs} \geq gmi\_nborder$$

$$\frac{j2\_gl - ju1\_gl + 1}{numLatProcs} \geq gmi\_nborder$$

and the number of processors to be requested to submit the executable is equal to *numWorkerProcs + 1*.

Here *i2_gl*, *i1_gl*, *j2_gl*, *ju1_gl*, and *gmi_nborder* are namelist variables (see Appendix F).

Appendix C

GMI NetCDF Files

We list all the NetCDF output files that the GMI code produces. For each file we provide:

- The list of variables that can be written out,
- The possible output frequencies of the data.

In Table C.1, we list the netCDF files that could be produced by the code.

File	Description
col	Column diagnostics at prescribed stations.
const	Main output file producing at least species concentrations.
aerDust	Mainly outputs optical depth for aerosols.
flux	File for fluxes.
freq#	Outputs selected variables at any specified frequency.
overpass#	Outputs selected variables that are produced at a prescribed time range.
qj	File for photolysis rate constants.
qk	File for thermal rate constants
qqjk	File for rates of photolytic and thermal processes.
restart	Restart file.
sad	File for surface area densitity related variables.
cloud	File containing cloud related variables.
tend	Tendencies file.

Table C.1: GMI netCDF files. Here # is 1, 2, 3, or 4.

All the output files contain at least the following variables (if necessary):

ai Hybrid pressure edge term

bi Hybrid sigma edge term

am Hybrid pressure term

bm Hybrid sigma term

pt

pressure Pressure (mb)

latdeg Latitude (deg)

londeg Longitude (deg)

mcor Grix box area (m^2)

nymd Current date (YYYYMMDD)

nhms Current time (HHMMSS)

Table C.2 gives a summary of the information contained in the netCDF files introduced in Section 4.3.2.

File	Default Suffix	Variables	Mean	Snapshot
col	.col.nc	psf		X
		humidity		X
		pot_temp		X
		const		X
const	.const.nc	const	X	X
		psf	X	X
		kel	X	X
		relHumidity	X	X
		gridBoxHeight	X	X
		tropopausePress	X	X
		potentialVorticity	X	
		overheadO3Col	X	X
		drydep		accum.
		wetdep		accum.
		semiss_out		accum.
		aerosolSurfEmiss		accum.
		emiss_3d_out		accum.
		aerosolEmiss3d		accum.
		lightning_no		accum.
		flashrate		accum.
		convectiveMassFlux		accum.
		mass	X	X
		metwater	X	X
		dms_oh	X	X
		dms_no3	X	X
		so2_oh	X	X
		so2_h2o2	X	X
		so2_o3	X	X

aeroDust	.aerodust.nc	optDepth	X	X
		aerosol column mass	X	X
		psf	X	X
		gridBoxHeight	X	X
overpass#	.overpass#.nc	const	X	X
		relHumidity	X	
		gridBoxHeight	X	
		tropopausePress	X	X
		mass	X	
		psf	X	X
		kel	X	X
		cloudOptDepth	X	X
		overheadO3Col	X	X
		metwater	X	
		qj	X	
		qqj	X	
		qqk	X	
freq#	.freq#.nc	const	X	
		const_column	X	X
		const_surface	X	X
		psf	X	X
		kel	X	X
		mass	X	X
		gridBoxHeight	X	X
		relHumidity	X	X
		metwater	X	X
		overheadO3col	X	X
		tropopausePress	X	X
		potentialVorticity	X	X
cloud	.cloud.nc	relHumidity	X	X
		psf	X	X
		kel	X	X
		mass	X	X
		gridBoxHeight	X	X
		cloudFraction	X	X
		liqWaterPath	X	X
		effectiveRadii	X	X
		cloudOpticalDepth	X	X
		totAerosolSulfate	X	X
		cloudDroplet	X	X
		cloudAlbedo	X	X
Mass Flux	.flux.nc	const_flux_x	X	X
		const_flux_y	X	X
		const_flux_z	X	X

		psf	X	
		amf	X	X
qj	.qj.nc	qj	X	X
		opt_depth	X	X
		O3+hv− >O1D	X	X
qk	.qk.nc	qk	X	X
qqjk	.qqjk.nc	qqj	X	X
		qqk	X	X
		yda		X
restart	.rst.nc	const		X
		h2ocond		X
		pctm2		X
sad	.sad.nc	sad	X	X
		hno3cond	X	X
		hno3gas	X	X
		h2oback	X	X
		h2ocond	X	X
		reffsts	X	X
		reffice	X	X
		vfall	X	X
tend	.tend.nc	ncumt		X

Table C.2: Content of each GMI netCDF output file.

For each of the files in Section 4.3.2, we provide in Table C.3 the frequency in which data can be written in the file.

File	Same Freq	Indepen Freq	Monthly	1st & 15th	Any freq	Last Step
const	X		X	X		X
aeroDust	X		X	X		X
cloud	X		X	X		X
qj	X		X	X		X
qk	X		X	X		X
qqjk	X		X	X		X
sad	X		X	X		X
tend	X		X	X		X
Mass Flux		X	X	X		X
overpass#		X	X	X		X
freq#		X	X	X	X	X
restart		X	X	X		X
col		X				

Table C.3: Frequency of GMI netCDF output files.

Appendix D

Important Features

D.1 From Species Indices to Species Names

In older versions of the code, setting namelist variables using species indices was not a simple task. When moving from one experiment to another, users were be interested in a given set of species but had to reset namelist variables matching species with their indices. In fact, their main concern was to know if a given species is part of a chemical mechanism, but not its index. Mistakes were unintentionally introduced, leading to a waste of time and computing resources. To alleviate these problems, we now use species labels instead of indices in the namelist file.

We wrote a Fortran module (*GmiSpeciesRegistry_mod* containing the function `getSpeciesIndex`) providing the species index given its name. The function `getSpeciesIndex` only takes a species name as its argument and does not depend on any chemical mechanism or any particular experiment. Two variables have to be passed to the module (will do internal initialization) for the function to work properly:

- *num_species*: total number of species used in the experiment
- *const_labels*: list of all species names used in the experiment.

Remark 8 *The module gets its information at run time but not at compilation time. This allows us to take into account tracer runs without making any specific provision in the code.*

We substituted old namelist variables (using species indices) with new ones. In the namelist file, we only need to provide the list of species names we are interested in and the code will figure out (using `getSpeciesIndex`) what indices they correspond to.

In Table D.1, we provide the list of the new namelist variables and the corresponding old ones. It is important to note that the old namelist variables remain part of the code (as regular variables) and are still used for calculations. The newly created variables are only local variables utilized to set the ones they replace in the namelist file.

Old Namelist variables	**New Namelist Variables**
nlGmiSpeciesConcentration SECTION	

fixed_const_map(1:n)	fixedConcentrationSpeciesNames
nlGmiDiagnostics SECTION	
pr_const_rec_flag(1:n)	concentrationSpeciesNames
pr_emiss_rec_flag(1:n)	surfEmissSpeciesNames
pr_drydep_rec_flag(1:n)	dryDepSpeciesNames
pr_wetdep_rec_flag(1:n)	wetDepSpeciesNames
pr_tend_rec_flag(1:n)	tendSpeciesNames
flux_species(1:n)	fluxSpeciesNames
ifreq#_species(1:n) (# = 1, 2, 3, 4)	freq#SpeciesNames
species_overpass#(1:n) (# = 1, 2)	overpass#SpeciesNames
noon_species(1:n)	noonSpeciesNames
local_species(1:n)	localSpeciesNames
col_diag_species(1:n)	colDiagSpeciesNames
nlGmiAdvection SECTION	
advec_flag(1:n)	advectedSpeciesNames
nlGmiEmission SECTION	
emiss_map(1:n)	emissionSpeciesNames
emiss_map_aero(1:n)	emissionAeroSpeciesNames
emiss_map_dust(1:n)	emissionDustSpeciesNames
nlGmiChemistry SECTION	
forc_bc_map(1:n)	forcedBcSpeciesNames

Table D.1: New namelist variables (and corresponding old ones) used to set species names instead of species indices in the namelist file.

We adopted the following principles for the new namelist variables:

1. For a given variable, the number of species names entered is no longer necessary.
2. Each variable is a long string that starts and ends with a single quote. Species names are separated with commas:
   `wetDepSpeciesNames = 'H2O2, HNO3, MP, N2O5',`
3. In the previous version of the code, `emiss_map` was set in the namelist file to determine the number of species in the emission input file and to select the species to be read in from the file. In the new setting, if a species is not read in, the name to be included in `emissionSpeciesNames` is xxx. The function will return -1 for the species index.
4. The order for entering the names is not important for diagnostics-related variables. However it is relevant for variables (`fixedConcentratinSpeciesNames, emissionSpeciesNames, emissionDustSpeciesNames, emissionAeroSpeciesNames, forcedBcSpeciesNames`) used to read in files.

5. Entering species names is not case sensitive. For example, the names `HNO3, hNO3, HnO3, HNo3, hnO3, hNo3, Hno3, hno3` correspond to the same species. Users can select any of these names to refer to `HNO3`.

6. If a species names does not exist, the code will abort.

Remark 9 *In previous versions of the GMI code, it was assumed that if a set of species is selected for constituent, wet deposition, dry deposition, and tendency output, the first species in the mechanism will be included by default. In this work, we did not make such an arrangement. Only the species provided by the user are considered for output.*

Here is an example of namelist setting for the AURA (combo, 124 species, no ship emission) experiments:

```
dryDepSpeciesNames    = 'CH2O, H2O2, HNO3, MP, N2O5, NO2, O3, PAN, PMN, PPN, R4N2',

wetDepSpeciesNames    = 'H2O2, HNO3, MP, N2O5',

surfEmissSpeciesNames = 'CH2O, CO, NO, ALK4, C2H6, C3H8, ISOP, MEK, PRPE',

fluxSpeciesNames      = 'CH2O, CH4, CO, HNO3, H2O2, MP, NO, NO2, N2O5, O3, PAN,
 SYNOZ',

freq1SpeciesNames     = 'CH4, CO, HNO3, N2O, O3, OH, ClO, Cl2O2, ClONO2, HCl, CFCl3,
 CF2Cl2',

overpass1SpeciesNames = 'CH2O, CO, NO, NO2, O3, OH',

colDiagSpeciesNames   = 'CH2O, CO, HNO2, HNO3, HNO4, H2O, HO2, H2O2, NO, NO2, NO3,
 N2O5, O3, OH, ALD2, ALK4, C2H6, C3H8, ISOP, PAN, PRPE, ACET',

tendSpeciesNames      = 'CH2O, CH4, CO, HNO3, H2O2, MP, NO, NO2, N2O5, O3, PAN, SYNOZ',

advectedSpeciesNames  = 'CH2O, CH4, CO, H2, HCOOH, HNO2, HNO3, HNO4, H2O2, MOH, MP,
 N2O, NO, NO2, NO3, N2O5, O3, Br, BrCl, BrO, BrONO2, HBr, HOBr, Cl, Cl2, ClO, Cl2O2,
 ClONO2 HCl, HOCl, OClO, CH3Br, CH3Cl, CH3CCl3, CCl4, CFCl3, CF2Cl2, CFC113, CFC114,
 CFC115, HCFC22, HCFC141b, HCFC142b, CF2Br2, CF2ClBr, CF3Br, H2402, ACTA, ALD2, ALK4,
 C2H6, C3H8, ETP, HAC, IALD, IAP, ISOP, MACR, MEK, MVK, PAN, PMN, PRPE, R4N2, RCHO,
 RCOOH, DEHYD, SYNOZ',

emissionSpeciesNames  = 'xxx, xxx, NO, NO, NO, NO, NO, CO, CO, CO, MEK, MEK, MEK, PRPE,
 PRPE, PRPE, C2H6, C2H6, C2H6, C3H8, C3H8, C3H8, ALK4, ALK4, ALK4, ALD2, ALD2, CH2O,
 CH2O, xxx, xxx, xxx, xxx, xxx, xxx',

forcedBcSpeciesNames  = 'CFCl3, CF2Cl2, CFC113, CFC114, CFC115, CCl4, CH3CCl3 HCFC22,
 HCFC141b, HCFC142b, CF2ClBr, CF2Br2, CF3Br, H2402, CH3Br, CH3Cl, CH4, N2O',
```

Remark 10 *The setting of each namelist variable can be done on a single line or on several lines. In case many lines are used, it is important to begin each line on the first or second column to avoid any problem.*

D.2 Station Diagnostics

In previous versions of the code, when we wanted to do station diagnostics we needed to set in the namelist file three variables :

col_diag_num: total number of stations

col_diag_site: complete list of stations

col_diag_lat_lon: locations (latitute/longitude) of stations.

Users were required not only to count the number of stations (can be several hundreds) but also to match the name of each station with its location. Removing/adding one station from the list involved the resetting of the above three variables with the possibility of mismatching. To facilitate the selection of stations, we :

- Constructed a file containing a list of all the identified stations and their locations. A namelist variable was created to point to the file. New stations can be added to the file at any time. The order of writting station information is irrelevant.
- Added a new namelist variable (long string) to enter the list of selected stations.
- Made changes in the code to check if each selected station exists in the file and then extract its location.
- Computed the number of selected stations at run time.

We created two new namelist variables (see Table D.2).

Old Namelist variables	**New Namelist Variables**
nlGmiDiagnostics SECTION	
col_diag_num	N/A
col_diag_site()	colDiagStationsNames
col_diag_lat_lon(2,n)	N/A
N/A	stationsInputFileName

Table D.2: New namelist variables for station diagnostics.

The piece of code used to carry out the above operations is:

```
! Construct the list of station using the long string
call constructListNames(col_diag_site, colDiagStationsNames)

col_diag_num = Count (col_diag_site(:) /= '')

if (col_diag_num /= 0) then
```

```
        do ic = 1, col_diag_num
           ! For each station in the list, check if it exists in the file
           ! and get its position (lat/lon)
           call getStationPosition(col_diag_site(ic), &
                                   col_diag_lat_lon(1,ic), &
                                   col_diag_lat_lon(2,ic), &
                                   stationsInputFileName)
        end do
         .
         .
         .
     end if
```

We present below a sample namelist setting for `colDiagStationsNames` and the first few lines of the file (`colDiagStationList.asc`) containing station information.

```
 colDiagStationsNames = 'SPO, MCM, HBA, FOR, NEU, SYO, PSA, MAR, MAQ,
  TDF, CRZ, LAU, CGO, ASP, CPT, EIC, JOH, REU, NAM, FIJ, TAH, CUI, SMO,
  PNA, WAT, ASC, NAT, SEY, BRA, MAL, NAI, SNC, CHR, KCO, PAR, TVD, PAN,
  VEN, RPB, GMI, POO, KUM, MLO, GTK, HON, TAI, ASK',
```

```
# ------------------------------------------------------------
# Station name       Lat     Lon                 Description
# ------------------------------------------------------------
SPO                -89.98   335.20
MCM                -77.83   166.60
HBA                -75.56   333.50
FOR                -71.00    12.00
NEU                -71.00   352.00
SYO                -69.00    39.58
PSA                -64.92   296.00
```

Remark 11 *While editing the file containing the station information, the following rules apply:*

1. *The first three lines of the file should start with the # character.*
2. *Column 1 to Column 16 are for the station name.*
3. *Column 17 to Column 25 are for the latitude of the station.*
4. *Column 26 to Column 34 are for the longitude of the station.*
5. *The remaining columns are reserved to describe stations and are not read in.*

D.3 Frequency Diagnostics

With the Freq files, users can select the variables they desire to output at any frequency:

1. *const* (only for selected species; written out if *freq#SpeciesNames* is set)
2. *constSurface* (only for selected species; written out if *freq#SpeciesNames* and *pr_const_surface_freq#* are set)
3. *constColTrop* (only for selected species; written out if *freq#SpeciesNames* and *pr_const_column_freq#* are selected only when using troposphere or combo mechanism)
4. *constColCombo* (only for selected species; written out if *freq#SpeciesNames* and *pr_const_column_freq#* are selected only when using the combo mechanism)
5. *kel* (written out if *pr_kel_freq#*=T)
6. *psf* (written out if *pr_psf_freq#*=T)
7. *metwater* (written out if *pr_metwater_freq#*=T)
8. *mass* (written out if *pr_mass_freq#*=T)
9. *relHumidity* (written out if *pr_rel_hum_freq#*=T)
10. *overheadO3Column* (written out if *pr_overheadO3col_freq#*=T)
11. *tropopausePressure* (written out if *pr_tropopausePress_freq#*=T)
12. *potentialVorticity* (written out if *pr_potentialVorticity_freq#*=T)

Here # is 1, 2, 3 or 4.

There are other Freq related namelist variables that are of interest:

```
     freq#_description: short description of the file
     freq#_name       : name of the Freq file

      pr_nc_freq#     : frequency for the outputs

Selected range of vertical levels
     k1_freq#         : lowest  level to output
     k2_freq#         : highest level to output

Selected horizontal domain for output
     lonRange_freq#(1): west  longitude between 0 and 360
     lonRange_freq#(2): east  longitude between 0 and 360
     latRange_freq#(1): south latitude  between -90 and 90
     latRange_freq#(2): north latitude  between -90 and 90
```

D.4 Overpass Diagnostics

Users can select a range of time for which they wish to save out selected variables and to write them in the file, <problem_name>.overpass#.nc (where # is 1, 2, 3 or 4). The variables are:

1. *const* (only for selected species; written out if *overpass#SpeciesNames* is set)
2. *kel* (written out if *do_mean*=T, and *pr_kel_overpass#*=T)
3. *psf* (written out if *do_mean*=T, and *pr_psf_overpass#*=T)
4. *metwater* (written out if *do_mean*=T, and *pr_metwater_overpass#*=T)
5. *qj* (written out if *do_mean*=T, and *pr_qj_overpass#*=T)
6. *qqj* (written out if *do_mean*=T, and *pr_qqjk_overpass#*=T)
7. *qqk* (written out if *do_mean*=T, and *pr_qqjk_overpass#*=T)

Four new namelist variables were added

pr_overpass# : if set to true, the overpass output file is produced. The file will still be created if any of the above conditions (namelist setting) is satistfied.

begTime_overpass# : begin time.

endTime_overpass# : end time.

pr_overpass#_period_days : overpass variable output period.

A sample namelist setting looks like (in the nlGmiDiagnostics section):

```
do_mean                  = T,
pr_overpass1             = T,
overpass1SpeciesNames    = 'CH2O, CO, NO, NO2, O3, OH',
begTime_overpass1        = 7.0d0,
begTime_overpass1        = 16.0d0,
pr_overpass1_period_days = -1.0d0,
```

In the above namelist setting, *pr_overpass1* = T. Therefore the file <problem_name>.overpass1.nc will be created. It will contain information (between 7am and 4pm) of the variables *const_overpass*, *kel_overpass*, and *psf_overpass* (the namelist variables *pr_const_overpass1*, *pr_kel_overpass1*, and *pr_psf_overpass1* will automatically be set to true). One can obtain the mean values of the variables *metwater_overpass*, *qj_overpass*, *qqj_overpass* and *qqk_overpass* by setting the namelist variables *pr_metwater_overpass1*, *pr_qj_overpass1*, *pr_qqj_overpass1*, and *pr_qqk_overpass1*, respectively, to true. It is important to note the following:

1. We have assumed that *do_mean*=T in order to be able to produce the overpass variables.

2. If *pr_overpass1* is not set at all in namelist file, the code will automatically set it to true if *overpass1SpeciesNames* has at least one species.

3. If *pr_overpass1* is set to false in the namelist file, then the file <problem_name>.overpass1.nc will not be created at all, regardless of the setting of the other variables.

D.5 Choice of Vertical Levels

All 3D variables are currently saved out on all vertical levels by default. However the user may only want to carry out post-processing analyses over a specific range of levels. At run time, the user can set the following namelist variables to select the range of vertical levels to output (this does not apply to the freq#, overpass#, and restart files):

pr_level_all : if set to T, all the vertical levels are considered, otherwise a range (selected in the next two variables) is used.

k1r_gl : First global altitude index for output. Should be at least equal to `k1_gl`.

k2r_gl : Last global altitude index for output. Should be at most equal to `k2_gl`.

A sample namelist setting looks like (in the nlGmiControl section):

```
pr_level_all =  F,
k1r_gl       =  3,
k2r_gl       = 20,
```

When `k1r_gl` and `k2r_gl` are set, they apply to all the netCDF output files except the restart file.

Appendix E

List of Species

We provide the list of the species used in the GMI code for the aerosol (A), GOCART aerosol (GA), micro_aerosol (MA), stratosphere (S), troposphere (T), and combo (C) chemical mechanisms.

Short Name	Unit	Long Name	Mechanism
ad	mol/mol		A/GA/MA
A3O2	mol/mol	Primary RO2 (C3H7O2) from propane	C/T
ACET	mol/mol	Acetone	C/T
ACTA	mol/mol	Acetic acid (C2H4O2)	C/T
ALD2	mol/mol	Acetaldehyde (C2H4O)	C/T
ALK4	mol/mol	C4,5 alkanes (C4H10)	C/T
ATO2	mol/mol	RO2 from acetone (C3H6O3)	C/T
B3O2	mol/mol	Secondary RO2 (C3H7O2) from propane	C/T
Br	mol/mol	Ground state atomic bromine (2P3/2)	C/S
BrCl	mol/mol	Bromine chloride	C/S
BrO	mol/mol	Bromine monoxide radical	C/S
BrONO2	mol/mol	Bromine nitrate	C/S
nOC	mol/mol		A/GA/MA
fOC	mol/mol		A/GA/MA
fBC	mol/mol		A/GA/MA
bOC	mol/mol		A/GA/MA
bBC	mol/mol		A/GA/MA
CF2Br2	mol/mol	Halon 1202	C/S
CF2ClBr	mol/mol	Halon 1211	C/S
CF3Br	mol/mol	Halon 1301	C/S
CFC113	mol/mol	CFC113 (C2Cl3F3)	C/S
CFC114	mol/mol	CFC114 (C2Cl2F4)	C/S
CFC115	mol/mol	CFC115 (C2ClF5)	C/S
C2H6	mol/mol	Ethane	C/T
C3H8	mol/mol	Propane	C/T
CCl4	mol/mol	Carbon tetrachloride	C/S
CFCl3	mol/mol	CFC11 (CFCl3)	C/S
CF2Cl2	mol/mol	CFC12 (CF2Cl2)	C/S
CH2O	mol/mol	Formaldehyde	C/T/S

CH3Br	mol/mol	Methyl bromide	C/S
CH3CCl3	mol/mol	Methyl chloroform	C/S
CH3Cl	mol/mol	Methyl chloride	C/S
CH4	mol/mol	Methane	C/T/S
Cl	mol/mol	Ground state atomic chlorine (2P3/2)	C/S
Cl2	mol/mol	Molecular chlorine	C/S
ClO	mol/mol	Chlorine monoxide radical	C/S
Cl2O2	mol/mol	Chlorine peroxide	C/S
ClONO2	mol/mol	Chlorine nitrate	C
CO	mol/mol	Carbon monoxide	C/T/S
fDMS	mol/mol		A/GA/MA
dust1	mol/mol		A/GA/MA
dust2	mol/mol		A/GA/MA
dust3	mol/mol		A/GA/MA
dust4	mol/mol		A/GA/MA
dust5	mol/mol		GA
EOH	mol/mol	Ethanol	C/T
ETO2	mol/mol	Ethylperoxy radical (C2H5O2)	C/T
ETP	mol/mol	Ethylhydroperoxide	C/T
GCO3	mol/mol	Hydroxy peroxyacetyl radical (C2H3O4)	C/T
GLYC	mol/mol	Glycoaldehyde (Hydroxyacetaldehyde C2H4O2)	C/T
GLYX	mol/mol	Glyoxal (2CHO)	C/T
GP	mol/mol	Peroxide (C2H4O4) from GCO3	C/T
GPAN	mol/mol	Peroxyacylnitrate (C2H3O6)	C/T
H	mol/mol	Ground state atomic hydrogen (2S)	C/S
H2	mol/mol	Molecular hydrogen	C/T/S
H24O2	mol/mol	Halon 24O2 (C2Br2F4)	C/S
H2O	mol/mol	Water	T
H2O2	mol/mol	Hydrogen peroxide	C/T/S/A/GA/MA
H2OCOND	mol/mol	Condensed water	C
HNO3COND	mol/mol	Condensed nitric acid	C
HAC	mol/mol	Hydroxyacetone (C3H6O2)	C/T/S
HBr	mol/mol	Hydrogen bromide	C/S
HCFC22	mol/mol	HCFC22 (CClF2H)	C/S
HCFC141b	mol/mol	HCFC141b (C2Cl2FH3)	C/S
HCFC142b	mol/mol	HCFC142b (C2ClF2H3)	C/S
HCl	mol/mol	Hydrochloric acid	C/S
HCOOH	mol/mol	Formic acid (CH3O2)	C/T
HNO2	mol/mol	Nitrous acid	C/T
HNO3	mol/mol	Nitric acid	C/T/S
HNO4	mol/mol	Pernitric acid	C/T
HO2	mol/mol	Perhydroxyl radical	C/T/S/A/GA/MA
HO2NO2	mol/mol		S
HOBr	mol/mol	Hypobromous acid	C/S
HOCl	mol/mol	Hypochlorous acid	C/S
IALD	mol/mol	Hydroxy carbonyl alkenes (C5H8O2) from isoprene	C/T
IAO2	mol/mol	RO2 (C5H9O8) from isoprene oxidation products	C/T
IAP	mol/mol	Peroxide (C5H10O5) from IAO2	C/T
INO2	mol/mol	RO2 (C5H8O3N) from ISOP+NO3	C/T
INPN	mol/mol	Peroxide (C5H8O6N2) from INO2	C/T
ISN1	mol/mol	RO2 (C4H7O4N) from ISN2	C/T

ISNP	mol/mol	Peroxide (C4H7O4N) from ISN1	C/T
ISOP	mol/mol	Isoprene	C/T
KO2	mol/mol	RO2 (C4H7O3) from C3 ketones	C/T
MACR	mol/mol	Methacrolein (C4H6O)	C/T
MAN2	mol/mol	RO2 (C4H6O6N) from MACR+NO3	C/T
MAO3	mol/mol	Peroxyacyl (C4H5O3) from MVK+MACR	C/T
MAOP	mol/mol	Peroxide (C4H6O3) from MAO3	C/T
MAP	mol/mol	Peroxyacetic acid (C2H4O3)	C/T
MCO3	mol/mol	Peroxyacetyl radical (C2H3O3)	C/T
MEK	mol/mol	Methyl ethyl ketone (C4H8O)	C/T
MGLY	mol/mol	Methylglyoxal (C3H4O2)	C/T
MO2	mol/mol	Methylperoxy radical (CH3O2)	C/T/S
MOH	mol/mol	Methanol	C/T
MP	mol/mol	Methyl hydroperoxide	C/T
MRO2	mol/mol	RO2 (C4H7O4) from MACR+OH	C/T
MRP	mol/mol	Peroxide (C4H8O4) from MRO2	C/T
MVK	mol/mol	Methyl vinyl ketone (C4H6O)	C/T
MVN2	mol/mol	C4H6O4N	C/T
N	mol/mol	Ground state atomic nitrogen	C/S
N2	m^-3	Molecular nitrogen	C/S
N2O	mol/mol	Nitrous oxide	C/S
N2O5	mol/mol	Dinitrogen pentoxide	C/T
NO	mol/mol	Nitric oxide	C/T/S
NO2	mol/mol	Nitrogen dioxide	C/T/S
NO3	mol/mol	Nitrogen trixide	C/T/S/A/GA/MA
O	mol/mol	Ground state atomic oxygen (3P)	C/S
O1D	mol/mol	First excited state of atomic oxygen (1D)	C/S
O2	m^-3	Molecular oxygen	C/S
O3	mol/mol	Ozone	C/T/S/A/GA/MA
OClO	mol/mol	Symmetrical chlorine dioxide	C/S
OH	mol/mol	Hydroxyl radical	C/T/S/A/GA/MA
PAN	mol/mol	Peroxyacetyl nitrate (C2H3NO5)	C/T
PMN	mol/mol	Peroxymethacryloyl nitrate (C4H5O5N)	C/T
PO2	mol/mol	RO2 (C3H7O3) from propene	C/T
PP	mol/mol	Peroxide (C3H8O3) from PO2	C/T
PPN	mol/mol	Prexypropionyl nitrate (C3H5O5N)	C/T
PRN1	mol/mol	RO2 (C3H5O5N) from propene+NO3	C/T
PRPE	mol/mol	Propene (C3H6)	C/T
PRPN	mol/mol	Peroxide (C3H6O3N) from PRN1	C/T
R4N1	mol/mol	RO2 (C4H9O3N) from R4N2	C/T
R4O2	mol/mol	RO2 (C4H9O2) from ALK4	C/T
R4P	mol/mol	Peroxide (C4H10O2) from R4O2	C/T
R4N2	mol/mol	C4-C5 alkylnitrates (C4H9O3N)	C/T
RA3P	mol/mol	Peroxypropyl alcohol (C3H8O2) from A3O2	C/T
RB3P	mol/mol	Peroxide from B3O2	C/T
RCHO	mol/mol	C2 aldehydes (C3H6O)	C/T
RCOOH	mol/mol	C2 organic acids	C/T
RCO3	mol/mol	Peroxypropionyl radical (C3H5O3)	C/T
RIO1	mol/mol	RO2 (C5H9O3) from isoprene oxidation products	C/T
RIO2	mol/mol	RO2 (C5H9O3) from isoprene	C/T
RIP	mol/mol	Peroxide (C5H10O3) from RIO2	C/T

ROH	mol/mol	C2 alcohols	C/T
RP	mol/mol	Methacrolein peroxy acid (C4H6O3)	C/T
sslt1	mol/mol		A/GA/MA
sslt2	mol/mol		A/GA/MA
sslt3	mol/mol		A/GA/MA
sslt4	mol/mol		A/GA/MA
fSO2	mol/mol		A/GA/MA
nSO2	mol/mol		A/GA/MA
fSO4a	mol/mol		A/GA
fSO4n1	mol/mol		A/GA
fSO4n2	mol/mol		A/GA
fSO4n3	mol/mol		A/GA
nSO4a	mol/mol		A/GA
nSO4n1	mol/mol		A/GA
nSO4n2	mol/mol		A/GA
nSO4n3	mol/mol		A/GA
SO4g	mol/mol		MA
SO4m1	mol/mol		MA
SO4n1	mol/mol		MA
SO4m2	mol/mol		MA
SO4n2	mol/mol		MA
SO4nOC	mol/mol		MA
SO4fOC	mol/mol		MA
SO4fBC	mol/mol		MA
SO4bOC	mol/mol		MA
SO4bBC	mol/mol		MA
SO4d1	mol/mol		MA
SO4d2	mol/mol		MA
SO4d3	mol/mol		MA
SO4d4	mol/mol		MA
SO4s1	mol/mol		MA
SO4s2	mol/mol		MA
SO4s3	mol/mol		MA
SO4s4	mol/mol		MA
SYNOZ	mol/mol	Synthetic ozone	C/T
VRO2	mol/mol	RO2 (C4H7O4) from MVK+OH	C/T
VRP	mol/mol	Peroxide (C4H8O4) from VRO2	C/T
Total Density	m^-3	Total	C/T/S
DEHYD			C/S
H2OAIR			C/S
H2OJETS	mol/mol	Aircraft-emitted water	C

Appendix F

Input Namelist Variables

A model run is controlled using a namelist input file named <problem_name>.in. The focus of this section is to describe the variables that constitute the namelist input. We provide the name of each variable, the default, and a brief description.

Variable Name	Type	Default Value	Description
nlGmiControl SECTION			
problem_name	C*128	'gmi_test'	The name of the problem to be run.
do_ftiming	L	F	Do fine timing?
Processor distribution:			
oneProcRun	L	F	Is this a one processor run?
numWorkerProcs	I	1	Number of worker prossors (numLonProcs * numLatProcs)
numLonProcs	I	1	Number of processors in the i direction (longitude).
numLatProcs	I	1	Number of processors in the j direction (latitude).
Global dimension info:			
gmi_nborder	I	4	Number of longitude and latitude ghost zones.
i1_gl	I	1	Index of first global longitude (no ghost zones).
i2_gl	I	72	Index of last global longitude (no ghost zones).
ju1_gl	I	0	Index of first global "u" latitude (no ghost zones).
jv1_gl	I	1	Index of first global "v" latitude (no ghost zones).
j2_gl	I	46	Index of last global "u&v" latitude (no ghost zones).
k1_gl	I	1	Index of first global altitude (no ghost zones).
k2_gl	I	29	Index of last global altitude (no ghost zones).
num_species	I	1	Number of species.
Time:			
leap_year_flag	I	0	Leap year flag: < 0: no year is a leap year $= 0$: leap years are determined normally > 0: every year is a leap year
begGmiTime	I	000000	Beginning hour/min/sec (HHMMSS).
begGmiDate	I	19890101	Beginning year/month/day (YYYYMMDD).
endGmiTime	I	010000	Ending hour/min/sec (HHMMSS).
endGmiDate	I	19890101	Ending year/month/day (YYYYMMDD).
gmi_sec	R	0.0	Total GMI seconds (s).
tdt	R	180.0	Model time step (s).
Main transport option:			
trans_opt	I	1	Transport option: 1: do LLNLTRANS transport

			2: do UCITRANS transport (non-parallel mode)
nlGmiMetFields SECTION			
General input data:			
gmi_data_dir	C*80	' '	Directory where input files are located.
Met data:			
met_opt	I	3	Met input option: 1: use values fixed in code for u, v, ps, kel; no other data set 2: read in a minimal set of met data: u, v, ps, & kel; no other data set 3: read in a full set of met data.
met_grid_type	C	'A'	Met grid type: 'A': use A grid (DAO, NCAR(CCM)) 'C': use C grid (GISS)
mdt	R	21600.0	Time increment for reading new met data; must be a multiple of tdt (s).
do_cycle_met	L	F	When the last met input file has been read, should the code cycle back and continue with the first file again?
do_timinterp_met	L	T	Should the met fields, except the winds, be timeinterpolated?
do_timinterp_winds	L	T	Should the wind fields be time interpolated? Note that pressure fields are always interpolated.
do_wind_pole	L	F	When met_opt = 1, should the transport be over the poles or around the equator?
met_infile_num	I	1	Index of NetCDF file to start reading met input data from.
mrnum_in	I	1	NetCDF file record to start reading met data from.
tmet1	R	0.0	Time tag of the mrnum_in (s).
do_read_met_list	L	F	Should the met file names be read in from met_filnam_list?
met_filnam_list	C*80	'met_filnam_list.in'	Name of file to get names of met input files from. Note that currently this file must reside in the same directory that you are running the gmi executable from.
met_infile_names()	C*128	' '	An array of met input file names (list may be used instead).
gwet_opt	I	0	Option for choosing which gwet variable to read (for GEOS4) 0: Read gwet1 1: Read gwettop
nlGmiSpeciesConcentration SECTION			
Base species concentration units = mixing ratio			
const_opt	I	2	Const input option:

			1: set const values to const_init_val 2: read in const values 3: solid body rotation 4: dummy test pattern with linear slope in each dimension 5: exponential in vertical (decays with height) 6: sin in latitude (largest at equator) 7: linear vertical gradient 8: sin in latitude (largest at equator) + vertical gradient
mw()	R	0.0	Array of species' molecular weights (g/mol).
const_init_val()	R	1.0d-30	When const_opt = 1, this array of values will be used to initialize each const species (note that if a negative const_init_val() marker is set in the namelist input file, all of the const_init_val's from negative value on will be set to the value preceding the negative value).
const_infile_name	C*128	' '	Constituent input file name.
const_var_name	C*32	'const'	NetCDF constituent variable name.
const_labels()	C*24	' '	Constituent string labels.
fixed_const_timpyr	I	12	Fixed const times per year: 1: one set of emissions per year (yearly) 12: twelve sets of emissions per year (monthly)
fixedConcentrationSpeciesNames	C*	"	List of fixed species concentration names as long string.
fixed_const_infile_name	C*128	' '	Fixed const input file name.
io3_num	I	0	Index of ozone constituent.
nlGmiDiagnostics SECTION			
ASCII output:			
Terminal screen output:			
pr_diag	L	F	Print some diagnostic output to screen?
pr_time	L	T	Should the time be printed to the terminal screen each time step (if false, will still get time output to the screen at the end of each day)?
Namelist file output:			
pr_nl	L	F	Should all the namelist variables be written to the file problem_name.nl?
Species/Mass ASCII file output:			
pr_ascii	L	T	Should the ASCII output file be written at all?
pr_ascii1	L	T	Should the first section of the ASCII output

			file be written (the mass data)?
pr_ascii2	L	F	Should the second section of the ASCII output file be written (the species concentration data)?
pr_ascii3	L	T	Should the third section of the ASCII output file be written (the species concentration min/maxs)?
pr_ascii4	L	F	Should the fourth section of the ASCII output file be written (total mass of each species)?
pr_ascii5	L	F	Should the fifth section of the ASCII output file be written (total production and loss of each species)?
ascii_out_n	I	1	Single species index to use.
ascii_out_i	I	1	Longitude index to use in the second section.
pr_ascii_step_interval	I	1	Interval for ASCII output: > 0: ASCII output at specified step interval $= -1$: ASCII output at monthly intervals
SmvgearII file output:			
pr_smv2	L	F	Should the SmvgearII output file be written (non-parallel mode only)?
General NetCDF output:			
pr_netcdf	L	T	Should any of the periodic output files be written at all?
hdr_var_name	C*32	'hdr'	NetCDF header variable name.
hdf_dim_name	C*32	'hdf_dim'	NetCDF header dimension name.
lat_dim_name	C*32	'latitude_dim'	NetCDF latitude dimension name.
lon_dim_name	C*32	'longitude_dim'	NetCDF longitude dimension name.
prs_dim_name	C*32	'pressure_dim'	NetCDF pressure dimension name.
spc_dim_name	C*32	'species_dim'	NetCDF species dimension name.
rec_dim_name	C*32	'rec_dim'	NetCDF record dimension name.
tim_dim_name	C*32	'time_dim'	NetCDF time dimension name.
pr_level_all	L	T	Should output be done on all the vertical levels? If pr_level_all=F, then set k1r_gl and k2r_gl
k1r_gl	I	1	First altitude index for output (k1r_gl $\geq$ k1_gl).
k2r_gl	I	29	Last altitude index for output (k2r_gl $\leq$ k2_gl).
pr_const	L	T	Should the periodic species concentrations output file be written?
pr_psf	L	F	Should the surface pressures be written to the periodic const output file?
pr_kel	L	F	Should the temperatures be written to the periodic const output file?
pr_mass	L	F	Should the mass be written to the periodic const output file?

pr_grid_height	L	F	Should the grid box height be written to the periodic const output file?
pr_relHumidity	L	F	Should the rel humidity be written to the periodic const output file?
pr_metwater	L	F	Should the meteorological water be written to the periodic const output file?
pr_dry_depos	L	F	Should the dry depositions be written to the periodic const output file?
pr_wet_depos	L	F	Should the wet depositions be written to the periodic const output file?
pr_surf_emiss	L	F	Should the surface emissions be written to the periodic const output file?
pr_overheadO3col	L	F	Should overhead ozone column be written out?
pr_tropopausePress	L	F	Should tropopause pressure be written out?
pr_potentialVorticity	L	F	Should potential vorticity be written out?
pr_qj	L	F	Should the periodic qj output file be written?
pr_qj_o3_o1d	L	F	Should the special reaction O3→O1D be saved with the qj's?
pr_qj_opt_depth	L	F	Should the optical depth be saved with the qj's?
pr_qk	L	F	Should the periodic qk output file be written?
pr_qqjk	L	F	Should the periodic qqjk output file be written?
pr_sad	L	F	Should the periodic sad output file be written?
pr_cloud	L	F	Should cloud related variables be written (for GT module)?
pr_tend	L	F	Should the periodic tendency diagnostics output file be written?
pr_const_all	L	T	Should all of the species concentrations be written to the periodic const output file?
pr_emiss_3d	L	F	Should 2d emissions be written to the periodic const output file?
pr_AerDust	L	F	Should the periodic aerosol/dust diagnostics be written?
do_aerocom	L	F	Should aerocom calculations be performed?
do_dust_emiss	L	F	
			> 0.0: periodic output at specified interval (days) -1.0: periodic output at monthly intervals -2.0: periodic output on 1st & 15th of each month
do_mean	L	F	Should means or current values be put in the periodic output files?
do_qqjk_inchem	L	F	If pr_qqjk is on, should qqj's & qqk's be determined inside the chemistry outside?
concentrationSpeciesNames	C*	"	List of species names for species concentration diagnostic as a long string.

			Note that concentrationSpeciesNames is only used if pr_const_all is false. Note that concentrationSpeciesNames is also used to determine the species written to dry_depos and wet_depos.
pr_nc_period_days	R	1.0	NetCDF output period: > 0.0: periodic output at specified interval (days) -1.0: periodic output at monthly intervals -2.0: periodic output on 1st & 15th of each month
pr_emiss_all	L	T	Should all the surface emissions be written to the periodic const output file? If set to F and pr_surf_emiss=T, then specify surfEmissionSpeciesNames.
surfEmissionSpeciesNames	C*	”	List of species names for surface emission diagnostic as a long string.
pr_drydep_all	L	T	Should all the dry depositions be written to the periodic const output file? If set to F and pr_dry_depos=T, then specify dryDepSpeciesNames.
dryDepSpeciesNames	C*	”	List of species names for dry_dep diagnostic as a long string.
pr_wetdep_all	L	T	Should all the wet depositions be written to the periodic const output file? If set to F and pr_wet_depos=T, then specify wetDepSpeciesNames.
wetDepSpeciesNames	C*	”	List of species names for wet_dep diagnostic as a long string.
pr_tend_all	L	T	Should periodic tendency diagnostics output file be written for all the species? If set to F and pr_tend=T, then specify tendSpeciesNames.
tendSpeciesNames	C*	”	List of species names for tendencies diagnostic as a long string.
`Flux Diagnostics:`			
pr_flux	L	F	Should the periodic flux diagnostics file be written?
fluxSpeciesNames(1:n)	C*16	”	List of species names used for flux diagnostics.
pr_const_flux	L	T	Should the periodic species concentrations output file be written?
pr_psf_flux	L	F	Should the surface pressure be written outin the flux file?
flux_name	C*8	’mf’	NetCDF flux variable name.
pr_nc_period_flux	R	1.0	flux output period: > 0.0: periodic output at specified interval (days) -1.0: periodic output at monthly intervals -2.0: periodic output on 1st & 15th of each month
`Overpass Output (# is 1, or 2):`			
pr_overpass#	L	F	Should the periodic overpass# output file be written?
overpass#SpeciesNames	C*	”	List species names for overpass# diagnostics as a long string.

pr_const_overpass#	L	F	Should the periodic species conc. be written for user defined species?
pr_psf_overpass#	L	F	Should surface pressure be written out?
pr_kel_overpass#	L	F	Should temperature be written out?
pr_qj_overpass#	L	F	Should photolysis rates be written out?
pr_qqjk_overpass#	L	F	Should photolysis rate constants be written out?
pr_metwater_overpass#	L	F	Should metwater be written out?
pr_totalMass_overpass#	L	F	Should mass be written out?
pr_relHumidity_overpass#	L	F	Should relative humidity be written out?
pr_gridBoxHeight_overpass#	L	F	Should grid box height be written out?
pr_cloudOptDepth_overpass#	L	F	Should cloud optical depth be written out?
pr_tropopausePress_overpass#	L	F	Should tropopause pressure be written out?
pr_overheadO3col_overpass#	L	F	Should overhead ozone column be written out?
begTime_overpass#	R	11.0	Beginning time for overpass#
endTime_overpass#	R	13.0	Ending time for overpass#
pr_overpass#_period_days	R	1.0	overpass netCDF output period: > 0.0: periodic output at specified interval (days) -1.0: periodic output at monthly intervals -2.0: periodic output on 1st & 15th of each month
`Frequency Output (# is 1, 2, 3, or 4):`			
pr_const_column_freq#	L	F	Should the periodic species conc. column file be written?
pr_const_surface_freq#	L	F	Should the periodic surf. species conc. file be written?
k1_freq#	I	k1	Minimum level for freq# output variables.
k2_freq#	I	k2	Maximum level for freq# output variables.
do_mean_freq#	L	F	
do_day1_freq#	L	F	
pr_freq#	L	F	Should the periodic output file at Frequency # be written?
pr_const_freq#	L	F	Should the periodic species conc. be written for user defined species?
pr_psf_freq#	L	F	Should surface pressure be written out?
pr_kel_freq#	L	F	Should temperature be written out?
pr_mass_freq#	L	F	Should mass be written out?
pr_rel_hum_freq#	L	F	Should relative humidity be written out?
pr_grid_height_freq#	L	F	Should grid box height be written out?
pr_overheadO3col_freq#	L	F	Should overhead ozone column be written out?
pr_potentialVorticity_freq#	L	F	Should potential vorticity be written out?

pr_tropopausePress_freq#	L	F	Should tropopause pressure pressure be written out?
pr_nc_freq#	R	1.0	
lonRange_freq#(1:2)	R	0.d0, 360.d0	Selected longitude range for outputs
latRange_freq#(1:2)	R	-90.d0, 900.d0	Selected latitude range for outputs
freq#SpeciesNames	C*	”	List species names for freq# diagnostics as a long string.
freq#_name	C*80	’ ’	
freq#_description	C*80	’ ’	Description to be included in the header of the nc file.
Column diagnostic NetCDF output:			
stationsInputFileName	C*128	”	File having a list of all possible stations (with their locations) for column disgnostics.
col_diag_period	R	3600.0	Column diagnostics output period (s).
colDiagStationsNames	C*	’ ’	List of selected stations (as a long string) for column diag.
colDiagSpeciesNames	C*	”	List of species names for column diagnostics as a long string.
col_diag_pres(1:10)	R	1000.0, ... , 100.0	Pressure levels for column diag. (mb).
nlGmiRestart SECTION			
pr_restart	L	F	Should a restart file be written?
do_overwrt_rst	L	T	Should the restart file be over-written?
pr_rst_period_days	R	7.0	Restart output period: > 0.0: restart output at specified interval (days) -1.0: restart output at monthly intervals -2.0: restart output on 1st & 15th of each month
rd_restart	L	F	Should a restart file be read?
restart_infile_name	C*128	’gmi.rst.nc’	Name of restart input file; note that currently this file must reside in the same directory that you are running the gmi executable from.
restart_inrec	I	last record # in rst file	Record number in restart (rst) input file to read from.
nlGmiAdvection SECTION			
advec_opt	I	1	Advection option: 0: no advection 1: do DAO2 advection
press_fix_opt	I	1	Pressure fixer option: 0: no pressure fixer used 1: LLNL pressure fixer used (Cameron-Smith)

			2: UCI pressure fixer used (Prather)
pmet2_opt	I	1	pmet2 option: 0: use pmet2 1: use (pmet2 - "global mean change in surface pressure")
advec_consrv_opt	I	2	Advection conserve option: 0: conserve tracer conc.; use pmet2 1: conserve tracer mass; use pmet2 2: conserve both tracer conc. & mass; use pctm2 Note that if press_fix_opt = 0 & advec_consrv_opt = 2, the code will generate an error and exit.
advec_flag_default	I	1	Set all species to do advection or not to do advection as the default; can then use advec_flag turn individual species either off, if the default is on; or on, if the default is off: 0: do not advect any species as default 1: advect all species as default
advectedSpeciesNames	C*	"	List of advected species names as a long string. Set advec_flag_default = 1 to use this variable.
j1p	I	3	Determines size of the Polar cap; j2p = j2_gl - j1p + 1
do_grav_set	L	F	Should gravitational settling of aerosols be done?
do_var_adv_tstp	L	F	Should variable advection time steps be taken as determined by the Courant condition?
nlGmiConvection SECTION			
Base convection units = kg/m^2*s			
convec_opt	I	0	Convection option: 0: no convection 1: do DAO2 convection 2: do NCAR convection
nlGmiDeposition SECTION			
Base deposition units = m/s			
do_drydep	L	F	Should dry deposition be done?
do_wetdep	L	F	Should wet deposition be done?
do_simpledep	L	F	Should simple deposition be done?
num_ks_sdep	I	1	Number of vertical layers to apply 2 day loss factor to in simple deposition.

wetdep_eff()	R	0.0	Wet deposition (scavenging) efficiencies; should be set to values between 0.0 and 1.0 for each species.
nlGmiDiffusion SECTION			
diffu_opt	I	0	Diffusion option: 0: no diffusion 1: do DAO2 vertical diffusion
vert_diffu_coef	R	0.0	Scalar vertical diffusion coefficient (m^2/s).
nlGmiEmission SECTION			
Base emissions units = kg/s			
emiss_opt	I	0	Emissions option: 0: no emissions 1: do LLNL emissions only 2: do LLNL + Harvard emissions
emiss_in_opt	I	0	Emissions input option: 0: no emissions data 1: set all emiss values to emiss_init_val 2: read in emiss values
emiss_conv_flag	I	0	Emissions conversion flag: 0: no conversion performed 1: use scalar emiss_conv_fac (scalar * kg/s => kg/s) 2: use calculated emissions conversion factor (kg/km^2*hr => kg/s)
semiss_inchem_flag	I	-1	Surface emissions inside chemistry flag: < 0: If emissions are on, surface emissions will be done in Smvgear chemistry if it is on; outside of chemistry if Smvgear chemistry is off. $= 0$: If emissions are on, surface emissions will be done outside of chemistry. > 0: If emissions are on, surface emissions will be done in Smvgear chemistry.
emiss_timpyr	I	1	Emission times per year: 1: one set of emissions per year (yearly) 12: twelve sets of emissions per year (monthly)
emissionSpeciesNames	C*	"	Ordered list of names of species (as a long string) to be read in from the emission file. If a species appears in the file but is but is not read in, it should be labeled 'xxx'.

emiss_conv_fac	R	1.0	Emission conversion factor when emiss_conv_flag = 1.
emiss_init_val	R	1.0	When emiss_opt = 1, this value will be used to initialize all emissions values.
emiss_infile_name	C*128	' '	Emissions input file name.
emiss_var_name	C*32	'emiss'	NetCDF emission variable name.
doReadDailyEmiss	L	F	Should we read the daily emission file?
begDailyEmissRec	I	1	beginning record for daily emission reading
endDailyEmissRec	I	1	ending record for daily emission reading
Harvard biogenic & soil emissions:			
isop_scale()	R	1.0d0	Isoprene scaling factors for each month.
Note that if ((emiss_opt == 2) && do_full_chem), the indices below will be automatically set by the setkin files.			
iacetone_num	I	0	Const array index for acetone (C3H6O) (ACET).
ico_num	I	0	Const array index for CO.
iisoprene_num	I	0	Const array index for isoprene (C5H8) (ISOP).
ipropene_num	I	0	Const array index for propene (C3H6) (PRPE).
ino_num	I	0	Const array index for NO.
fertscal_infile_name	C*128	"	Fertilizer scale infile name.
lai_infile_name	C*32	"	Leaf area index infile name.
light_infile_name	C*128	"	Light infile name.
precip_infile_name	C*128	"	Precipitation infile name.
soil_infile_name	C*128	"	Soil type infile name.
veg_infile_name	C*128	"	Vegetation type infile name.
isopconv_infile_name	C*128	"	Isoprene conversion infile name.
monotconv_infile_name	C*128	"	Monoterpene conversion infile name.
MEGAN Emissions:			
doMEGANemission	L	F	Should we do MEGAN emissions?
laiMEGAN_InfileName	C*128	' '	AVHRR leaf-area-indices infile name.
aefMboMEGAN_InfileName	C*128	' '	Annual emission factor for methyl butenol infile name.
aefIsopMEGAN_InfileName	C*128	' '	Annual emission factor for isoprene infile name.
aefMonotMEGAN_InfileName	C*128	' '	Annual emission factor for monoterpenes infile name.
aefOvocMEGAN_InfileName	C*128	' '	Annual emission factor for other biogenic VOCs infile name.
Ship Emission:			
do_ShipEmission	L	F	Should we do ship emission calculation?

Galactic Cosmic Ray:			
do_gcr	L	F	Should Galactic Cosmic Ray source of N and NO be turned on?
gcr_infile_name	C*128	' '	Input file for Galactic Cosmic Ray source parameters
Emission Scaling Factors:			
doScaleNOffEmiss	L	F	Should we use fossil fuel scaling factors?
scFactorNOff_infile_name	C*128	' '	Scaling factor for NO fossil fuel emission infile name.
doScaleNObbEmiss	L	F	Should we use biomass burning scaling factors?
scFactorNObb_infile_name	C*128	' '	Scaling factor for biomass burning emission infile name.
Michigan aerosol and dust emissions:			
emiss_aero_opt	I	0	aerosol emission option 0: no aerosol emission 1: Michigan aerosol emissions 2: GOCART aerosol emissions
naero	I	0	number of aerosol emissions.
emissionAeroSpeciesNames emiss_aero_infile_name	C* C*128	" ' '	Ordered list of names of aerosol species (as a long string) to be read in from the aerosol emission file. Name of file containing michigan aerosol emissions.
emiss_dust_opt	I	0	0,1; 0 fo no michigan dust emissions.
emiss_dust_opt	I	0	dust emission option 0: no dust emission 1: Michigan dust emissions 2: GOCART dust emissions
ndust	I	0	number of dust emissions.
nst_dust	I	1	number of starting point in time for michigan dust emissions.
nt_dust	I	1	number of times of dust emissions per michigan dust emissions file.
emissionDustSpeciesNames emiss_dust_infile_name	C* C*128	" ' '	Ordered list of names of dust species (as a long string) to be read in from the dust emission file. Name of file containing michigan dust emissions.
GOCARTerod_infile_name	C*128	"	
GOCARTocean_infile_name	C*128	"	
GOCARTerod_mod_infile_name	C*128	"	
nlGmiChemistry SECTION			
chem_opt	I	0	Chemistry option: 0: no chemistry (age of air, etc.)

			1: call Radon/Lead chemistry 2: call SmvgearII 3: call simple loss (N2O, etc.) 4: call forcing boundary condition for a tracer (CO2, etc.) 5: call Synoz tracer (if num_species=1 then just Synoz, if num_species=2 then Nodoz tracer is species number 2) 6: call Beryllium chemistry 7: call Quadchem 8: call Sulfur chemistry
cloudDroplet	I	1	cloud droplet option: 1: Boucher and LohMan Correlation 2: Nenes and Seinfeld Parameterization 3: Abdul-Razzak and Ghan Parameterization 4: Segal amd Khain Correllation
chem_cycle	R	1.0	Number of time steps to cycle chemistry calls on: < 1.0: chemistry will subcycle $= 1.0$: chemistry called each time step
chem_mask_klo	I	k1_gl	Lowest grid level at which chemistry is calculated.
chem_mask_khi	I	k2_gl	Highest grid level at which chemistry is calculated.
loss_opt	I	0	Stratospheric loss option 0: do not use stratospheric loss 1: use stratospheric loss in gmi_step.F
oz_eq_synoz_opt	I	0	conversion of syzoz to ozone option 0: no conversion 1: do conversion
synoz_threshold	R	Huge	Chemistry turned off where synoz > this threshold (mixing ratio).
t_cloud_ice	R	263.0	Temperature for cloud ice formation.
do_chem_grp	L	F	Should chemical groups be used?
do_smv_reord	L	F	Should the grid boxes be reordered in order of stiffness?
do_wetchem	L	F	Should wet chemistry be done?
Aerosol/Dust Calculations For trop and combo without interactive aerosols and chemistry			
AerDust_var_name	C*32	' '	netCDF aerosol/dust variable name
AerDust_infile_name	C*128	' '	aerosol/dust input file name

do_AerDust_Calc	L	F	Should you do aerosol/dust calculations?
AerDust_Effect_opt	I	0	Radiative effects or/and heteregeneous chemistry 0: rad. effects on and het. chem. on 1: rad. effects off and het. chem. on 2: rad. effects on and het. chem. off 3: rad. effects off and het. chem. off
Be-7/Be-10 chemistry:			
be_opt	I	1	Beryllium star table option: 1: use Koch table for Be-7 and Be-10 2: use Nagai tables for Be-7 and Be-10
t_half_be7	R	53.3d0	Half life of Beryllium-7, or other cosmogenic radionuclide (days).
t_half_be10	R	5.84d8	Half life of Beryllium-10, or other cosmogenic radionuclide (days).
yield_be7	R	4.5d-7	Yield factor for Beryllium-7, or other cosmogenic radionuclide (unitless).
yield_be10	R	2.5d-7	Yield factor for Beryllium-10, or other cosmogenic radionuclide (unitless).
Base forcing boundary condition units = mixing ratio			
forc_bc_opt	I	1	Forcing boundary condition option: 1: set all forc_bc values to forc_bc_init_val 2: read in forc_bc 3: calculate forc_bc
fbc_j1	I	ju1_gl	Forcing boundary condition j1 (low latitude).
fbc_j2	I	j2_gl	Forcing boundary condition j2 (high latitude).
forc_bc_years	I	1	Number of years of forcing data.
forc_bc_start_num	I	1	Forcing boundary condition start number; index for year to use.
forc_bc_kmin	I	1	Minumum k level for forcing boundary condition.
forc_bc_kmax	I	1	Maximum k level for forcing boundary condition.
forcedBcSpeciesNames	C*	”	Ordered list of species names (as a long string) used for forcing boundary condition.
forc_bc_init_val	R	0.0	When forc_bc_opt = 1, this value will be used to initialize all forc_bc values (ppmv).
forc_bc_incrpyr	R	0.3	Forcing boundary condition emission increase per year.
forc_bc_lz_val	R	0.0	Value to which lower zones are forced.
forc_bc_infile_name	C*128	'forc_bc_co2.asc'	Forcing boundary condition input file name.
Base simple loss units = s^{-1}			
loss_freq_opt	I	1	Loss frequency option:

			1: set all loss_freq values to loss_init_val 2: read in loss data 3: use NCAR loss
kmin_loss	I	k1_gl	Minimum vertical index at which loss will occur; currently, below this altitude a constant boundary condition is enforced using const_init_val for all species.
kmax_loss	I	k2_gl	Maximum vertical index at which loss will occur.
loss_init_val	R	0.0	When loss_freq_opt = 1, this value will be used to initialize all loss_freq values.
loss_data_infile_name	C*128	'loss_n2o.asc'	Loss data input file name.
Surface Area Density (SAD):			
sad_opt	I	0	Surface area density (SAD) option: 0: do not allocate or process SAD array 1: allocate, but zero out SAD array 2: call Considine code (i.e., Condense) 3: read SAD array from a file of monthly averages
sad_var_name	C*32	'sad'	NetCDF sad variable name.
sad_dim_name	C*32	'sad_dim'	NetCDF sad dimension name.
h2oclim_opt	I	2	Water climatology input option: 1: set all h2oclim values to h2oclim_init_val 2: read in h2oclim
h2oclim_timpyr	I	12	Water climatology times per year 1: yearly 12: monthly
ch4clim_init_val	R	0.0	When h2oclim_opt = 1, this value will be used to initialize all ch4clim
h2oclim_init_val	R	0.0	When h2oclim_opt = 1, this value will be used to initialize all h2oclim values.
h2oclim_infile_name	C*128	' '	Water climatology input file name.
lbssad_opt	I	2	Liquid binary sulfate input option: 1: set all lbssad values to lbssad_init_val 2: read in lbssad
lbssad_timpyr	I	12	Liquid binary sulfate times per year: 1: yearly 12: monthly
lbssad_init_val	R	0.0	When lbssad_opt = 1, this value will be used

			to initialize all lbssad values.
lbssad_infile_name	C*128	' '	Liquid binary sulfate input file name.
qk / qqk:			
qk_var_name	C*32	'qk'	NetCDF qk variable name.
qqk_var_name	C*32	'qqk'	NetCDF qqk variable name.
qk_dim_name	C*32	'qk_dim'	NetCDF qk dimension name.
qqk_dim_name	C*32	'qqk_dim'	NetCDF qqk dimension name.
Reaction rate adjustment:			
do_rxnr_adjust	L	F	Adjust reaction rates?
rxnr_adjust_infile_name	C*128	' '	Reaction rate adjustment input file name.
rxnr_adjust_var_name	C*32	'reac_rate_adj'	NetCDF reaction rate adjustment variable name.
nlGmiPhotolysis SECTION			
Base photolysis/qj units = s^{-1}			
phot_opt	I	1	Photolysis option: 0: no photolysis 1: set all qj values to qj_init_val 2: read in qj values 3: use a version of fastj (to be used with fastj_opt) 4: lookup table for qj (Kawa style) 5: lookup table for qj (Kawa style) + use ozone climatology for column ozone calc. 6: calculate from table and GMI data (Quadchem) 7: read in qj values (2-D, 12 months)
fastj_opt	I	0	fastj option (set together with phot_opt=3): 0: for fastj 1: for fast_JX 2: for fast_JX53b 3: for fast_JX53c
cross_section_file	C*128	' '	X-Section quantum yield input file name
rate_file	C*128	' '	Master input file name
T_O3_climatology_file	C*128	' '	T & O3 climatology input file name
scattering_data_file	C*128	' '	Aerosol/cloud scattering data input file name Only used for fast_JX53b and fast_JX53c
do_ozone_inFastJX	L	F	Should ozone columns be computed inside fast_JX?

			By default fast_JX uses the model ozone columns.
do_clear_sky	L	T	Should clear sky photolysis be done?
fastj_offset_sec	R	0.0d0	Offset from model time at which to do fastj (s).
qj_init_val	R	1.0d-30	When phot_opt = 1, this value will be used to initialize all qj values.
qj_infile_name	C*128	' '	qj input file name.
qj_var_name	C*32	'qj'	NetCDF qj variable name.
qqj_var_name	C*32	'qqj'	NetCDF qqj variable name.
qj_dim_name	C*32	'qj_dim'	NetCDF qj dimension name.
qqj_dim_name	C*32	'qqj_dim'	NetCDF qqj dimension name.
Surface albedo:			
sfalbedo_opt	I	0	Surface albedo option: 0: no sfalbedo 1: set each type of sfalbedo to an intial value 2: read in monthly sfalbedo values from a netCDF file 3: read in values of four types of surface albedo from the met data
saldif_init_val	R	0.1	Surface albedo for diffuse light (near IR); when sfalbedo_opt = 1, this value will be used to initialize all saldif values.
saldir_init_val	R	0.1	Surface albedo for direct light (near IR); when sfalbedo_opt = 1, this value will be used to initialize all saldir values.
sasdif_init_val	R	0.1	Surface albedo for diffuse light (uv/vis); when sfalbedo_opt = 1, this value will be used to initialize all sasdif values.
sasdir_init_val	R	0.1	Surface albedo for direct light (uv/vis); when sfalbedo_opt = 1, this value will be used to initialize all sasdir values.
sfalbedo_infile_name	C*128	' '	Surface albedo input file name.
Solar Cycle:			
do_solar_cycle	L	F	Should solar cycle for incoming radiation be turned on? (currently works with lookup table only)
sc_infile_name	C*128	' '	file for solar cycle coefficients
UV albedo:			
uvalbedo_opt	I	0	UV albedo option: 0: no uvalbedo 1: set all uvalbedo values to uvalbedo_init_val 2: read in monthly uvalbedo values from an ASCII file

			3: read in bulk surface albedo values from the met data
uvalbedo_init_val	R	0.1	When uvalbedo_opt = 1, this value will be used to initialize all uvalbedo values.
uvalbedo_infile_name	C*128	' '	Uvalbedo input file name.
nlGmiTracer SECTION			
tracer_opt	I	0	Tracer run option: 0: no tracer 1: tracer run
efold_time	R	0.0	e-folding time of the tracer (in days)
tr_source_land	R	0.0	land source for the tracer
tr_source_ocean	R	0.0	ocean source for the tracer
nlGmiLightning SECTION			
lightning_opt	I	0	Lighning option: 0: lightning NO emissions read from file 1: parameterized lightning 2: no lightning
i_no_lgt	I	0	Index to the location of NO_lgt in the emiss infile
desired_g_N_prod_rate	R	5.0	global nitrogen production rate (in Tg.)

Table F.1: Namelist variables